UNIVERSITY OF CALIFORNIA,
IRVINE

Password-Based Cryptographic Protocols in the Client-Server Setting

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Jiayu Xu

Dissertation Committee:
Professor Stanislaw Jarecki, Chair
Professor Michael Goodrich
Professor Alice Silverberg

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my PhD advisor, Stanislaw Jarecki. It was Stas who got me started in the wonderful world of cryptography. He has always impressed me with his creativity, patience, and kindness. I always enjoy my discussions with him, from the high-level ideas to the most trivial details. I hope that this dissertation serves as a record of my work with Stas so far, and that we will have fruitful collaborations in the future.

I am very grateful to Hugo Krawczyk, who has guided me throughout my Ph.D. years and has been a role model to me. Hugo showed me the importance of diligent work; I'll never forget the countless days and nights working with him to finish a paper. He taught me how to explain ideas clearly, and how to present my work to others. Also, I am grateful for Hugo's help in my search for an academic position after my Ph.D. work.

Thanks to Tatiana Bradley, my long-time colleague and collaborator. I appreciate Tatiana's skill and speed in understanding theoretical concepts, and I am thankful to her deep insights on our work. It is always a pleasure working with Tatiana, and I wish her all the best in her future career.

This dissertation would not have been possible without the guidance and help from my co-authors and friends: Jan Camenisch, Karthik Gajulapalli, Jung Yeon Hwang, Aggelos Kiayias, Taekyoung Kwon, Joohee Lee, Anja Lehmann, Gregory Neven, Ji Sun Shin, and Boyang Wei. Thank you all!

Last but not least, I would like to express my deep gratitude to my parents, who have supported me with love for so many years. While acknowledging that life as a Ph.D. student can be challenging, they have never questioned my choice, and did their best to help me overcome all kinds of difficulties. Throughout my life, they encouraged me to think with an open mind, to try new things, and to pursue what I am truly interested in. I am extremely fortunate to be their son.

CURRICULUM VITAE

Jiayu Xu

EDUCATION

Doctor of Philosophy in Computer Science **2019**
University of California, Irvine *Irvine, CA, USA*

Masters of Science in Computer Science **2015**
University of California, Irvine *Irvine, CA, USA*

Bachelor of Science in Information Sciences **2013**
Peking University *Beijing, China*

RESEARCH EXPERIENCE

Graduate Research Assistant **2015–2019**
University of California, Irvine *Irvine, CA, USA*

TEACHING EXPERIENCE

TA for Discrete Mathematics for Computer Science (ICS 6D) **Fall 2018**
University of California, Irvine *Irvine, CA, USA*

TA for Discrete Mathematics for Computer Science (ICS 6D) **Spring 2018**
University of California, Irvine *Irvine, CA, USA*

TA for Applied Cryptography (CS 167) **Winter 2018**
University of California, Irvine *Irvine, CA, USA*

TA for Applied Cryptography (CS 167) **Winter 2017**
University of California, Irvine *Irvine, CA, USA*

TA for Computational Linear Algebra (ICS 6N) **Fall 2017**
University of California, Irvine *Irvine, CA, USA*

REFEREED CONFERENCE PUBLICATIONS

T. Bradley, S. Jarecki, and J. Xu:
Strong asymmetric PAKE based on trapdoor CKEM **2019**
International Cryptology Conference (CRYPTO) (to appear)

T. Bradley, J. Camenisch, S. Jarecki, A. Lehmann, G. Neven, and J. Xu:
Password-authenticated public-key encryption **2019**
Applied Cryptology and Network Security (ACNS) (to appear)

J. Y. Hwang, S. Jarecki, T. Kwon, J. Lee, J. S. Shin, and J. Xu:
Round-reduced modular construction of asymmetric password-authenticated key exchange [42] **2018**
Security and Cryptography for Networks (SCN)

S. Jarecki, H. Krawczyk, and J. Xu:
OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks [46] **2018**
Theory and Applications of Cryptographic Techniques (EUROCRYPT)

S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu:
TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF [45] **2017**
Applied Cryptology and Network Security (ACNS)

S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu:
Highly-efficient and composable password-protected secret sharing (or: how to protect your bitcoin wallet online) [44] **2016**
IEEE European Symposium on Security and Privacy (EuroS&P)

ABSTRACT OF THE DISSERTATION

Password-Based Cryptographic Protocols in the Client-Server Setting

By

Jiayu Xu

Doctor of Philosophy in Computer Science

University of California, Irvine, 2019

Professor Stanislaw Jarecki, Chair

Passwords have become the most ubiquitous form of client-server authentication on the Internet nowadays. Password-over-TLS, the almost universal password authentication protocol in practice, suffers from two major drawbacks: (i) it requires a secure channel; and (ii) the server sees the client's password in the clear.

This dissertation focuses on another approach of password authentication, which eliminates the two shortcomings above. We study cryptographic protocols in the password-only setting, that is, the only information shared between the client and the server is the short and low-entropy password. We present highly efficient realizations of two kinds of such protocols: (1) Password-Protected Secret Sharing (PPSS), in which the client stores a long secret (e.g., its private key) in a group of servers, and recovers the secret via interacting with a subset of servers using a password; and (2) asymmetric Password-Authenticated Key Exchange (aPAKE), in which the client (which enters a password) and the server (which stores a password file) establish the same secret key. All these protocols are resilient to man-in-the-middle attacks (i.e., no authenticated channel is required) as well as server compromise: the only forms of attacks are the unavoidable ones, namely online password guessing attacks and offline dictionary attacks in case of server compromise.

We present thorough description of our protocols, the proofs of their security, and analyses

of their computational costs. All security proofs are in the Universally Composable (UC) framework, which addresses subtle vulnerabilities of passwords (non-uniform distribution over the dictionary, reuse of the same password over different accounts, etc.) in a natural and easy-to-argue way, and thus is preferred over the traditional game-based security model.

Chapter 1

Introduction

1.1 Problem Statement

Password-over-TLS. Passwords constitute the most ubiquitous form of authentication on the Internet, from the most mundane to the most sensitive applications. The almost universal password authentication method in practice relies on TLS/SSL and consists of the client sending its password to the server under the protection of a client-to-server confidential TLS channel. At the server, the password is decrypted and verified against a one-way image typically computed via hash iterations applied to the password and a random "salt" value. Both the password image and salt are stored for each client in a so-called "password file." In this way, an attacker who succeeds in compromising the server and stealing the password file is forced to run an exhaustive *offline dictionary attack* to find clients' passwords given a set ("dictionary") of candidate passwords. The two obvious disadvantages of this approach are: (i) the password appears in cleartext at the server during login; and (ii) security breaks if the TLS channel is established with a compromised server's public key (a major concern

given today's too-common Public-Key Infrastructure (PKI) failures[1]).

Password-Authenticated Key Exchange (PAKE) and its weaknesses. Password protocols have been extensively studied in the cryptographic literature – including in the above client-server setting where the client is assumed to possess an authentic copy of the server's public key [41], but the main focus has been on *password-only* protocols where the client does not need to rely on any outside keying material (such as public keys). The basic setting is modeled as *Password-Authenticated Key Exchange (PAKE)*, which considers two parties that share the same low-entropy password with the goal of establishing shared session keys secure against *offline dictionary attacks*, namely, against an active attacker that possesses a small dictionary from which the password has been chosen. The only viable option for the attacker should be *online guessing attacks*, where the adversary runs the prescribed PAKE protocol on a password guess with either the client or the server, and succeeds if its guess was correct. While such attack is unavoidable, its effect can be reduced by limiting the number of unsuccessful authentication session each party is willing to run.

The PAKE security model was introduced by Bellovin and Merritt [12] and was formalized by Bellare *et al.* [10] and Boyko *et al.* [19] via a game-based definition, and then by Canetti *et al.* [26] in the Universally Composable (UC) framework [24]. The UC definition of PAKE has become the de facto standard in the cryptographic literature on PAKEs because it is widely recognized as capturing several security issues pertinent to PAKEs which the game-based PAKE notions of [10, 19] do not cover. Specifically, apart of standard UC guarantee of security under arbitrary protocol composition, UC PAKE implies *forward security*, i.e., security of past protocol sessions in case of password compromise, and security for *arbitrary password distribution*, which implies security for *password mistyping*

[1]PKI failures include stealing of server private keys, software that does not verify certificates correctly, clients that accept invalid or suspicious certificates, certificates issued by rogue Certificate Authority (CA)'s, servers that share their TLS keys with others – e.g., CDN providers or security monitoring software, information (including passwords) that traverses networks in plaintext form after TLS termination, and more. For an overview of this topic, see e.g., [60].

and for *related passwords*.

Most of cryptographic PAKE literature focuses on the *symmetric* PAKE setting, where both parties hold the same password. However, if the client-to-server password authentication was implemented with a symmetric PAKE, a compromise of the server would leak the passwords of all clients who authenticate to that server.

T-PAKE, PPSS, and aPAKE protocols. In order to overcome the weakness of PAKE protocols in the event of server compromise as described above, we study two types of PAKE extensions which are resilient to some form of server compromise.

- In a (t, n)-*Threshold PAKE (T-PAKE)* protocol, the single authentication server is replaced by a group of n servers, and no information on passwords is leaked (i.e., offline dictionary attacks are eliminated) even if up to t servers are compromised.
  T-PAKE was introduced by Mackenzie *et al.* [63]. Bagherzandi *et al.* [8] proposed a related notion of *Password-Protected Secret Sharing (PPSS)*, which simplifies the notion of T-PAKE by reducing the goal of key exchange between client and servers to that of the client retrieving (in the reconstruction phase) a single secret previously shared with the servers (in the initialization phase). Due to the low-overhead generic PPSS-to-T-PAKE compiler [8, 43], the design of T-PAKE's is essentially reduced to the design of PPSS. On the other hand, PPSS is also an important primitive in its own right, allowing for online storage of sensitive information like keys, credentials, or personal records, with availability and privacy protection. The only token needed for retrieving stored information is a *single* password, and both information and password remain private if no more than t servers are compromised (and if the adversary does not guess or learn the password).

- In an *asymmetric PAKE (aPAKE)* (a.k.a. *augmented* or *verifier-based* PAKE)

protocol, the server stores a password file, which consists of a one-way image of the password, as in the aforementioned password-over-TLS approach. Hence, upon a server compromise and the stealing of the password file, an attacker is forced to perform an exhaustive offline dictionary attack as in password-over-TLS. No other attack, except for an inevitable online guessing attack, should be feasible.

The first formalization of aPAKE was introduced by Bellovin and Merrit [13] and formalized in the game-based approach by Boyko *et al.* [19]. Subsequently, Gentry *et al.* [40] extended the UC PAKE model of [26] to the case of an adaptive server compromise, and forcing the adversary to stage an offline dictionary attack to recover the password after such compromise. While several aPAKE protocols were proven secure in game-based models, some argued only informally, e.g., [19, 62, 61, 4, 21, 14, 54], the UC aPAKE notion is stronger than game-based aPAKE for the same reasons that UC PAKE notion is stronger than game-based PAKE, thus ideally we would like to know protocols which realize the UC aPAKE notion of [19] and are comparable in efficiency and cryptographic assumptions to standard authenticated key agreement protocols used in TLS. However, not much is known about provably secure UC aPAKE's.

Weaknesses of aPAKE protocols. A common deficiency of all aPAKE protocols in the literature, including those being proposed for practical use and regardless of their underlying formalism, is that they are *all vulnerable to pre-computation attacks*. Namely, the attacker can *pre-compute* a table of values based on a password dictionary D, so as soon as it succeeds in compromising a server it can *instantly* find a client's password. This weakens the benefits of security against server compromise that motivate the aPAKE notion in the first place. Moreover, while current definitions require that the attacker cannot exploit a server compromise without incurring a workload proportional to the dictionary size $|D|$, these definitions allow all this workload to be spent *before* the actual

server compromise happens. Indeed, this weakness in the existing UC aPAKE security definition [40] is needed to accommodate aPAKE protocols that store a one-way *deterministic* mapping of the client's password at the server, say $H(\mathsf{pw})$. Such protocols trivially fall to a pre-computation attack as the attacker can build a table of $(H(\mathsf{pw}), \mathsf{pw})$ pairs for all $\mathsf{pw} \in D$, and once it compromises the server, it finds the value $H(\mathsf{pw})$ associated with a client and immediately, in $\log(|D|)$ time, finds that client's password. Such attack can be mitigated by "personalizing" the password map, e.g., hashing the password together with the user ID. This forces the attacker to pre-compute separate tables for individual clients, yet all this effort can still be spent before the actual server compromise.

Note that the standard password-over-TLS protocol prevents pre-computation by hashing passwords with a random salt visible to the server only. In contrast, existing aPAKE protocols that do not rely on PKI, either do not use salt or if they do, the salt is transmitted from server to client during login *in the clear*.[2] Given that password stealing via server compromise is the main avenue for collecting billions of passwords by attackers, the above vulnerability of existing aPAKE protocols to pre-computation attacks is a serious flaw, and in this aspect password-over-TLS is more secure than all known aPAKE protocols.

1.2 Our Contributions

In this study, we present highly efficient PPSS and aPAKE protocols proven secure in the UC framework, and we propose *strong aPAKE (saPAKE)*, a new notion of aPAKE which eliminates pre-computation attacks described above. We list our specific contributions below:

[2] Note that even if the aPAKE protocol runs over TLS, the transmitted salt is open to a straightforward active attack. An additional weakness introduced by the use of public salt is enabling "username enumeration attacks" that help attackers identify clients and targets – see [30].

- In Chapter 3 we present TOPPSS, a simple PPSS protocol with remarkable and hard-to-beat performance. The reconstruction procedure requires *just 1 exponentiation per server and a total of 2 (multi-)exponentiations for the client* (independent of the number of servers), plus $O(t)$ modular multiplications by each party. Communication is also optimal: The client sends a single group element to a subset of $t + 1$ servers and gets one group element from each server. Furthermore, we present a UC notion of PPSS, which can be seen as a significant relaxation of the UC PPSS notion of [22] (called T-PASS therein), and show that TOPPSS satisfies this UC notion under the One-More Diffie-Hellman (OMDH) assumption.

- In Chapter 4 we present a new compiler which convert any UC secure *symmetric* PAKE protocol into a UC secure *asymmetric* PAKE protocol. It adds only a *single additional message* to the underlying PAKE; moreover, this single extra message is sent from client to server, and therefore in an application where the PAKE instance, which establishes a secure session key for both parties, is followed by an explicit client-to-server entity authentication, e.g., the client uses the session key output by PAKE to send a MAC on the PAKE transcript to the server, this additional message can be piggybacked with the client's explicit entity authentication flow. Likewise, if the last message of the symmetric UC PAKE is client-to-server, our compiler also adds no additional communication flow to the protocol. We show that the compiler satisfies the UC notion of aPAKE under the Computational Diffie-Hellman (CDH) assumption.

- In Chapter 5 we initiate the study of *strong aPAKE (saPAKE)* protocols that strengthen the aPAKE security notion by *disallowing pre-computation attacks.* We first formalize this notion in the UC framework by modifying the aPAKE functionality from [40]. We then present two generic constructions. The first builds the saPAKE protocol from any UC aPAKE protocol (namely one that satisfies the original definition from [40]) so that one can "salvage" existing aPAKE protocols.

The second builds the saPAKE protocol from any regular Authenticated Key Exchange (AKE) protocol resilient to "Key-Compromise Impersonation (KCI)" attacks. Both constructions satisfies our UC notion of saPAKE under the OMDH assumption. Finally, we provide a highly efficient instantiation of our second transformation, which we call the *OPAQUE protocol*. OPAQUE combines the best properties of existing aPAKE protocols and of the password-over-TLS protocol. As any secure aPAKE protocol, it offers two fundamental advantages over the TLS-based solution: It does not rely on PKI, and the plaintext password is never in the clear at the server. The only way for an attacker that observes (or actively controls) a session at a server to learn the password is via an exhaustive offline dictionary attack; watching or participating in a session with the client does not help the attacker. At the same time, OPAQUE resolves the major flaw of existing aPAKE protocols relative to password-over-TLS, namely, their vulnerability to pre-computation attacks.

Chapter 2

Preliminaries

2.1 Notations

Throughout this study, we use the following notational conventions:

- If a and b are strings, then $[a||b]$ denotes their concatenation;
- If D is a set, then $|D|$ denotes its cardinality;
- ":=" denotes the computation of a deterministic function, while $\leftarrow$ denotes the computation of a randomized algorithm;
- For an integer n, we write $n{+}{+}$ as an abbreviation for $n := n + 1$, and $n{-}{-}$ as an abbreviation for $n := n - 1$;
- If D is a set, then $x \leftarrow_{\mathrm{R}} D$ denotes the procedure of picking x uniformly at random from D;
- If E is an event, then $\Pr[E]$ denotes its probability;

- κ denotes the security parameter (presented in its unary form 1^κ while given as an input). The adversary against a primitive is always given 1^κ as an input, so we omit it.

We also list the abbreviations which appear in this study in Table 2.1 for readers' reference.

Abbreviation	Stands for
AKE	Authenticated Key Exchange
aPAKE	Asymmetric Password Authenticated Key Exchange
CDH	Computational Diffie-Hellman (Assumption)
IC	Ideal Cipher
KCI	Key-Compromise Impersonation
KE	Key Exchange
MAC	Message Authentication Code
OMDH	One-More Diffie-Hellman (Assumption)
OPRF	Oblivious Pseudorandom Function
PKI	Public-Key Infrastructure
PPSS	Password-Protected Secret Sharing
ROM	Random Oracle Model
saPAKE	Strong Asymmetric Password Authenticated Key Exchange
T-OMDH	Threshold One-More Diffie-Hellman (Assumption)
T-OPRF	Threshold Oblivious PseudoRandom Function
T-PAKE	Threshold Password Authenticated Key Exchange
UC	Universally Composable

Table 2.1: List of abbreviations

2.2 Cryptographic Assumptions and Primitives

We briefly review the cryptographic assumptions and primitives used in this study.

The CDH assumption. Let $\mathbb{G}$ be a cyclic group with g as a generator and m as its order. We assume that m is a prime with $|m|$ polynomial in κ. The *Computational Diffie-Hellman (CDH) assumption* in $(\mathbb{G}, g, m)$ states that for any efficient algorithm $\mathcal{A}$,

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{CDH},\mathbb{G}} = \Pr_{a,b \leftarrow_{\mathrm{R}} \mathbb{Z}_m}[\mathcal{A}(g, g^a, g^b) = g^{ab}]$$

is a negligible function of κ.

In other words, given two challenge values g^a and g^b which are random group elements, $\mathcal{A}$ cannot compute g^{ab} except with negligible probability.

The (Gap) OMDH assumption. The (N, Q)*-One More Diffie-Hellman (OMDH) assumption* [9] in a group $(\mathbb{G}, g, m)$ states that for any efficient algorithm $\mathcal{A}$,

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{OMDH},\mathbb{G}} = \Pr_{k \leftarrow_{\mathrm{R}} \mathbb{Z}_m, g_1, \ldots, g_N \leftarrow_{\mathrm{R}} \mathbb{G}}[\mathcal{A}^{(\cdot)^k}(g, g^k, g_1, \ldots, g_N) = S]$$

is a negligible function of κ, where $S = \{(g_{j_s}, g_{j_s}^k) | s = 1, \ldots, Q+1\}$, Q is the number of $\mathcal{A}$'s queries to the $(\cdot)^k$ oracle which on input $a \in \mathbb{G}$ outputs a^k, and $j_s \in \{1, \ldots, N\}$ for $s \in \{1, \ldots, Q+1\}$.

In other words, suppose $\mathcal{A}$ has access to a "k-th power" oracle, to which the number of queries is limited by Q. $\mathcal{A}$ is given N random elements in $\mathbb{G}$ as the challenge values. Since $\mathcal{A}$ is allowed to query the $(\cdot)^k$ oracle Q times, it is able to compute the k-th power of any Q of the N challenge values $g_1, \ldots, g_N$. The assumption postulates that $\mathcal{A}$ cannot compute

the k-th power of any $Q+1$ of the N challenge values (i.e., compute the k-th power of "one more" challenge value) except with negligible probability.

Note that CDH is equivalent to $(1,0)$-OMDH.

The (N,Q)-Gap OMDH assumption is the same with the (N,Q)-OMDH assumption, except that the adversary $\mathcal{A}$ is additionally given access to a DDH oracle in $\mathbb{G}$, $\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)$, which on inputs $a,b,c,d \in \mathbb{G}$ outputs 1 if $\log_a b = \log_c d$ and 0 otherwise. Formally, the (N,Q)-Gap OMDH assumption in a group $(\mathbb{G},g,m)$ states that for any polynomial-time algorithm $\mathcal{A}$,

$$\mathbf{Adv}^{\text{Gap-OMDH},\mathbb{G}}_{\mathcal{A}} = \Pr_{k\leftarrow_{\mathrm{R}}\mathbb{Z}_m, g_1,\ldots,g_N\leftarrow_{\mathrm{R}}\mathbb{G}}[\mathcal{A}^{(\cdot)^k,\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)}(g,g^k,g_1,\ldots,g_N) = S]$$

is a negligible function of κ, where S, Q, $(\cdot)^k$ and j_s are the same as in the definition of the (N,Q)-OMDH assumption.

Authenticated encryption. An *authenticated encryption scheme* AE is a tuple of efficient algorithms $(\mathsf{KeyGen},\mathsf{AuthEnc},\mathsf{AuthDec})$, where

- $\mathsf{KeyGen}(1^\kappa)$ outputs a key k (where $|k| \geq \kappa$);
- $\mathsf{AuthEnc}_k(m)$ outputs a ciphertext c;
- $\mathsf{AuthDec}_k(c)$ outputs a message m. Without loss of generality, we assume that $\mathsf{AuthDec}$ is a deterministic algorithm.

The correctness property of AE requires that for any κ and any message m in the message space, if $k \leftarrow \mathsf{KeyGen}(1^\kappa)$ and $c \leftarrow \mathsf{AuthEnc}_k(m)$, then $m = \mathsf{AuthDec}_k(c)$.

The security of AE states that for any efficient algorithm $\mathcal{A}$,

$$\mathbf{Adv}^{\text{SEC},\mathsf{AE}}_{\mathcal{A}} = |\Pr[\mathcal{A}^{\mathsf{AuthEnc}_k(\cdot)}(c_0) \text{ outputs } 1] - \Pr[\mathcal{A}^{\mathsf{AuthEnc}_k(\cdot)}(c_1) \text{ outputs } 1]|$$

is a negligible function of κ, where $k \leftarrow \mathsf{KeyGen}(1^\kappa)$, $m_0, m_1 \leftarrow \mathcal{A}^{\mathsf{AuthEnc}_k(\cdot)}$, $c_0 \leftarrow \mathsf{AuthEnc}_k(m_0)$, and $c_1 \leftarrow \mathsf{AuthEnc}_k(m_1)$.

In other words, consider two security games $\mathbf{G}_0$ and $\mathbf{G}_1$. In $\mathbf{G}_b$ ($b \in \{0,1\}$), $\mathcal{A}$ first outputs two messages m_0 and m_1, and then receives c_b, an encryption of m_b. Throughout the game, $\mathcal{A}$ is given access to the encryption oracle $\mathsf{AuthEnc}_k(\cdot)$. The security property states that $\mathbf{G}_0$ and $\mathbf{G}_1$ are indistinguishable in $\mathcal{A}$'s view.

The authenticity of AE states that for any efficient algorithm $\mathcal{A}$,

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{AUTH},\mathsf{AE}} = \Pr[\mathcal{A}^{\mathsf{AuthEnc}_k(\cdot)} \text{ outputs } c \text{ s.t. } m := \mathsf{AuthDec}_k(c) \neq \perp \text{ and } m \notin Q]$$

is a negligible function of κ, where $k \leftarrow \mathsf{KeyGen}(1^\kappa)$ and Q is the set of all $\mathcal{A}$'s queries to the $\mathsf{AuthEnc}_k(\cdot)$ oracle.

In other words, suppose that $\mathcal{A}$ is given access to the encryption oracle $\mathsf{AuthEnc}_k(\cdot)$. Then $\mathcal{A}$ is able to generate some valid ciphertexts (i.e., ciphertexts that do not decrypt to $\perp$) via querying the encryption oracle. The authenticity property states that $\mathcal{A}$ cannot generate another valid ciphertext except with negligible probability.

2.3 Security Models and Frameworks

The random oracle model. All security results in this study are proven in the *Random Oracle Model (ROM)*. For a description of ROM and its implementation, see [11].

The UC framework. All security results in this study are in the *Universal Composability (UC) framework*, proposed by Canetti in [24]. Here we provide a very high-level and informal overview of the UC framework based on [25]; for a detailed description, we refer to [24].

The UC framework is an example of *simulation*-based security definition. In this paradigm, the security notion is captured by a trusted party (called the *functionality*) which interacts with the protocol participants and an *ideal adversary*; the input-output behaviors of these protocol participants and the ideal adversary are specified in the description of the functionality (hence the attacking ability of the ideal adversary is clearly defined). We say that a protocol is secure if the real adversary cannot gain more information than the ideal adversary which merely interacts with the functionality. Looking a bit closer, consider a *real world* where there is an adversary against the protocol, and an *ideal world* where the adversary interacts with the functionality. We say that the protocol is secure if for any efficient real adversary $\mathcal{A}$, there is an efficient ideal adversary (also called the *simulator*) SIM which generates a view indistinguishable to $\mathcal{A}$'s real-world view (i.e., SIM "simulates" $\mathcal{A}$'s real-world view).

The traditional simulation-based security model only provides security guarantee for a *stand-alone* execution of the protocol. In contrast, UC security implies that the protocol remains secure even when *arbitrarily composed* with other protocols, including insecure ones. In order to model such composed protocols which might be insecure, another adversarial party called the *environment* is added. The environment specifies the inputs for all protocol participants, receives all outputs, and interacts with the adversary in an arbitrary way. A protocol is UC secure (also called *realizes the UC functionality*) if for any efficient environment $\mathcal{Z}$ and any efficient real adversary $\mathcal{A}$, there is an efficient simulator SIM which generates a view indistinguishable to $\mathcal{Z}$'s real-world view.

The UC theorem states the follows. Consider a UC functionality $\mathcal{F}$ and a protocol Π in which all participants have access to $\mathcal{F}$ (as a trusted party). Let ρ be any protocol which UC realizes $\mathcal{F}$, and Π^ρ be the composed protocol where interactions with $\mathcal{F}$ are replaced with participating in an instance of ρ. Then Π and Π^ρ have the same input-output behavior; in particular, if Π UC realizes a functionality $\mathcal{G}$, then Π^ρ also UC realizes $\mathcal{G}$.

Chapter 3

TOPPSS: Cost-minimal Password-Protected Secret Sharing Based on Threshold OPRF

In this chapter we present TOPPSS, the most efficient PPSS protocol to date – and using the PPSS-to-T-PAKE compiler of [43] also the most efficient T-PAKE – with a hard-to-beat complexity. Informally, a (t, n)-PPSS protocol, as formulated in the PKI-free setting by [43], allows a client to share a random secret s among n servers under the protection of its password pw such that (1) a reconstruction protocol involving at least $t + 1$ honest servers recovers s if the client inputs the (correct) password pw; (2) the compromise of up to t servers leaks no information about either s or pw; (3) an adversary who corrupts $t' \leq t$ servers and has q_{C} interactions with the client and q_{S} interactions with the uncorrupted servers can test at most $\frac{q_{\mathrm{S}}}{t-t'+1} + q_{\mathrm{C}}$ passwords. (In the PKI setting one can set $q_{\mathrm{C}} = 0$.)

Our starting point is the PPSS protocol of Jarecki *et al.* [43], the most efficient PPSS protocol to date, without PKI authentication in reconstruction, with a reconstruction phase

that takes a single round (2 messages) between a client and each server and only costs 2 (multi-)exponentiations per server for the client and roughly 2 (multi-)exponentiations per each participating server. [43] used a game-based definition of PPSS security adopted from [8] to the password-only setting (i.e., no PKI assumption). We modify and improve the PPSS protocol in [43] in several ways:

- *Highly efficient performance:* We reduce the computational cost of the protocol in [43] to *a single exponentiation per server and a total of 2 exponentiations for the client* (independent of the number of servers).

- *Non-reliance on PKI and secure channels:* A main accomplishment of the PPSS protocols of [22] and [43] is the non-reliance on secure channels or PKI during the reconstruction phase. This is a major benefit since PPSS assumes a client that only knows its user ID and password, and does not carry auxiliary devices with authenticated information. Moreover, the increasing vulnerabilities of certificate-based authentication translate into weaknesses in protocols that rely on such authentication. Fortunately, we achieve our optimal performance while still dispensing with the need of secure channels or PKI (except for a trusted initialization phase needed in all PPSS protocols).

- *UC security:* In addition to the significant performance improvement relative to [43], we also improve on their security analysis by providing a proof that our protocol satisfies a UC formalization of PPSS. This formalization is in itself a significant contribution of our work. Indeed, while a UC definition of PPSS appeared in the work of Camenisch *et al.* [22], our formulation significantly relaxes this functionality in a way that enables the proof of a much more efficient protocol. To obtain this relaxed UC functionality (and the UC proof of our protocol) we utilize a *ticketing mechanism*, used e.g., by [43] in their formalism of V-OPRF (see below), which allows us to dispense with the need to extract the client's input (in this case, the client's password) during an execution

of a UC PPSS protocol, something that requires heavier cryptographic mechanisms and results in higher performance costs, as with the protocol of [22]. The ticketing mechanism ensures that in order to test a single password guess, the attacker must impersonate the client to $t+1$ servers or impersonate $t+1$ servers to the client, which is optimal in terms of security against guessing attacks and constitutes the very essence of the PPSS security notion. By relaxing the UC functionality for PPSS, we allow for much more efficient realizations, while maintaining the security of the PPSS-to-T-PAKE compiler.

The results presented in this chapter are based on the work published in [44] and [45], with significant revision of the security proofs.

3.1 Overview

Oblivious Pseudorandom Function (OPRF). A central ingredient in the protocol of [43] that we preserve in our solution is the notion of an *Oblivious Pseudorandom Function (OPRF)* [37]. Roughly speaking, an OPRF is a protocol between two parties, one (the server as in the context of application to PPSS) holding a key k for a PRF F and one (the client) holding an input x, where no party learns anything except for the input holder who learns $F_k(x)$. This notion has been shown to be useful in many different contexts [37, 47], but defining it so that it can be implemented inexpensively is non-trivial. For example, using an Multi-Party Computation (MPC)-type definition would require a costly implementation to achieve concurrent security (as needed here) and would require secure channels (undesired here). To resolve this problem, [43] introduced a UC-based OPRF definition that allowed them to build a PPSS protocol with concurrent security and without secure channels for reconstruction. Moreover, their use of a "ticketing mechanism" in their OPRF definition (which is also similar in spirit to e.g., blind signature definitions,

cf. [7]) allowed them to obtain very efficient instantiations by avoiding extractable proofs of knowledge or similar costly mechanisms (this is the ticketing mechanism that, as mentioned before, we have borrowed for our own UC formulation of PPSS). At the same time, in order for the OPRF to fit their PPSS protocol, [43] strengthened the security notion of OPRF adding a *verifiability* property that allows clients to detect dishonest behaviors of the PPSS servers during reconstruction. Unfortunately, this additional property introduces the need to use zero-knowledge proofs in the implementation of their OPRF, costing one multi-exponentiation for the client and server in each client-server interaction and leads to an increase in the amount of communication as well.

As our first observation to reduce the PPSS computational costs, we resolve this problem by relaxing the *Verifiable* OPRF (V-OPRF) notion of [43] into a plain UC OPRF functionality that does not provide verifiability, and therefore enables an optimal implementation without zero-knowledge proofs at all (it also has the potential of better fitting other OPRF applications). We note that forgoing the verifiability property weakens the *robustness* of our PPSS solution, namely, the ability to discard incorrect computations during reconstruction. Yet, we can enjoy the best of the two worlds: We can run the highly efficient protocol without zero-knowledge proofs, and only resort to the zero-knowledge proofs in case the reconstruction fails. Thus, in the normal case of a non-adversarial run the cost of zero-knowledge is saved. Finally, we remark that in real-world applications we expect n to be a small number, in which case checking different subsets of $t + 1$ servers until finding a non-corrupted subset is a practical approach that completely dispenses with zero-knowledge proofs.

Threshold Oblivious Pseudorandom Function (T-OPRF). Our second observation is that the OPRF in the protocol of [43] can be replaced with its *threshold* (or multi-party) counterpart which we define as *Threshold OPRF (T-OPRF)*. We provide a UC definition

of T-OPRF as a functionality that allows a group of n servers to secret-share a key k for PRF F with a shared PRF evaluation protocol which lets the client compute $F_k(x)$ on its input x, such that both x and k are secret if no more than t of n servers are corrupted. T-OPRF can be viewed as an input-oblivious strengthening of Distributed PRF (DPRF) of Naor *et al.* [64], hence in particular T-OPRF can replace DPRF in all its applications, e.g., for corruption-resilient Key Distribution Center, and long-term information protection (see [64]). We design a T-OPRF protocol, which we call 2HashTDH. This T-OPRF protocol is essentially a "threshold exponentiation" protocol, where each server computes m^{k_i} on input m where k_i is the server's secret-share of the PRF key k.

TOPPSS: PPSS based on T-OPRF. Using the strong notion of T-OPRF security above we show TOPPSS, a compiler which transforms UC T-OPRF into UC PPSS at negligible additional cost in the Random Oracle Model (ROM). We prove that TOPPSS realizes UC PPSS under the following assumptions in ROM. Let $t' \leq t$ denote the number of parties actually controlled by an attacker. First, our results imply that in the so-called *full corruption* case, i.e., if $t' = t$, the same (Gap) One-More Diffie-Hellman (OMDH) assumption used in [43] implies that the attacker must query one uncorrupted party per each input on which the attacker wants to obtain the function value. Since this is the case where the attacker controls the full threshold t of servers it is also the case for any $t' < t$. In the application to PPSS this means that the attacker can test up to $q_S + q_C$ passwords, which matches the $\frac{q_S}{t-t'+1} + q_C$ bound for $t' = t$. Since many existing works on T-PAKE, e.g., [63, 33, 20, 51, 3, 75], implicitly assume the $t' = t$ case by defining security using the simplified $q_S + q_C$ bound on the number of passwords the adversary can test, we call this level of security a *standard threshold security* for T-PAKE/PPSS.

Secondly, for the general case of $t' \leq t$, we show that TOPPSS achieves the stronger $\frac{q_S}{t-t'+1}+q_C$ bound assuming a generalization of the OMDH assumption which we call (Gap) *Threshold*

One-More Diffie-Hellman (T-OMDH). As a sanity check for the T-OMDH assumption we show that the T-OMDH problem is hard in the generic group model in Appendix A. Since OMDH is a special case of T-OMDH, to the best of our knowledge this is also the first generic group analysis of OMDH. The stricter bound implies that an adversary controlling $t' \leq t$ servers must contact $t-t'+1$ uncorrupted servers for each input on which it wants to compute the function, which coincides with the standard threshold security notion when $t' = t$, but it is stronger for $t' < t$. For example, it means that the default network adversary who does not corrupt any party but runs q sessions with each server, can test up to $\frac{qn}{t+1}$ passwords, whereas the standard threshold security would in this case upper-bound the number of tested passwords only by qn.

As a point of comparison we consider a generic compiler from *any* OPRF to T-OPRF. This compiler performs multi-party computation of the server code in the underlying OPRF protocol, but in the case of the OPRF of Section 3.2 such MPC protocol has the same low computational cost as the customized T-OPRF protocol 2HashTDH discussed above, i.e., 1 exponentiation per server and 2 for the client, with the only drawback of adding an additional communication round to enforce an agreement between the servers on the client's input to the MPC protocol. On the other hand, since the security depends only on the basic OPRF, the resultant two-round T-OPRF protocol achieves the $\frac{q_S}{t-t'+1} + q_C$ bound based solely on OMDH for all $t' \leq t$.

Related work. The first T-PAKE by Mackenzie *et al.* [63] required ROM in the security analysis and relied on PKI, namely, it assumed that the client can validate the public keys of the servers *during the reconstruction phase.*[1] Gennaro and Raimondo [33] dispensed with ROM and PKI (in authentication) but increased protocol costs. Abdalla *et al.* [3] showed a

[1]When we say that PPSS/T-PAKE assumes PKI we mean that it relies on it for the security of the reconstruction/authentication phase. By contrast, the *initialization phase* of any PPSS/T-PAKE solution must assume some trusted infrastructure, e.g., PKI, or otherwise each party could be initializing the protocol with an impostor.

PKI-free T-PAKE in ROM with fewer communication rounds than the T-PAKE of [63], but the client establishes a key with only one designated *gateway* server. Yi *et al.* [75] showed a similar round-reduction without ROM. The case of $n = 2$ servers, known as 2-PAKE, received special attention starting with Brainard *et al.* [20, 73] on 2-PAKE in ROM and PKI, and several works [51, 54, 16, 55] addressed the non-PKI and non-ROM case. Still, each of these T-PAKE protocol requires server-to-server communication. If communication is mediated by the client then the lowest round complexity is 3 for $n > 2$ [3] and 2 for $n = 2$ [16, 55].

Bagherzandi *et al.* [8] introduced the notion of PPSS with the goal of simplifying T-PAKE protocols. Specifically, they showed a PPSS protocol in ROM assuming PKI, with 2 rounds, constant-sized messages, and $8(t+1)$ (multi-)exponentiations per client, and a low-cost PKI-model compiler from PPSS to T-PAKE. Camenisch *et al.* [22] constructed another PPSS protocol, called T-PASS (for *Threshold Password-Authenticated Secret Sharing*), without assuming PKI but with $14n$ exponentiations for the client, 7 exponentiations per server, and 5 rounds of communication.

Jarecki *et al.* [43] showed significantly faster PPSS protocols, also without assuming PKI: The PPSS of [43] takes a single round (two messages) between a client and each server, and uses 2 (multi-)exponentiations per server and $2t + 3$ (multi-)exponentiations for the client, secure under (Gap) OMDH in ROM. (They also showed a 4-message non-ROM PPSS with $O(n \cdot |\mathsf{pw}|)$ exponentiations using Paillier encryption.) In related works, [23] showed a single-round *proactive* PPSS in the PKI setting for the case of $t = n$, and [5] showed general methods for ensuring robustness in PPSS reconstruction, and a non-ROM PPSS using $O(|\mathsf{pw}|)$ exponentiations in a prime-order group.

Another important aspect of these PPSS solutions is the type of security notion they achieve. Both the PKI-model PPSS notion of [8] and the PKI-free PPSS notion of [43] were game-based, while [22] provided UC definitions of the PPSS functionality. The essence of the UC

PPSS definition is that the only attack the adversary can stage is the inevitable one, namely, an online dictionary attack where validating a single password guess requires interaction with either $t+1$ instances of the servers or with the client. The UC definitions have further advantages for a password-based notion like PPSS, as discussed in Section 1.1.

3.2 The OPRF Functionality $\mathcal{F}_{\text{OPRF}}$ and Its Realization

Here we introduce our UC functionality, $\mathcal{F}_{\text{OPRF}}$, presented in Figure 3.1. It is derived from the $\mathcal{F}_{\text{VOPRF}}$ functionality of [43] by stripping off the "verifiability" properties of the latter. Specifically, $\mathcal{F}_{\text{VOPRF}}$ allows a client to check consistency between different runs of the OPRF; namely, that each time that the function is run with the same server on the same input, the same answer is received (otherwise the client aborts). This requires servers to have public keys and requires the OPRF implementation to involve zero-knowledge proofs. By omitting the verifiability condition we simplify the OPRF definition, implementation and applicability.

In the description below, we assume that $\mathsf{P} \in \{\mathsf{C}, \mathcal{A}^*\}$.
Public Parameter: PRF output-length ℓ, polynomial in security parameter κ.
For every server S, initialize $\mathsf{tickets}(\mathsf{S})$ to 0.

- On $(\mathsf{Eval}, sid, \mathsf{S}, x)$ from P, record $\langle \mathsf{P}, \mathsf{S}, x \rangle$ and send $(\mathsf{Eval}, sid, \mathsf{P}, \mathsf{S})$ to $\mathcal{A}^*$.
- On $(\mathsf{SvrComplete}, sid)$ from S, set $\mathsf{tickets}(\mathsf{S})$++ and send $(\mathsf{SvrComplete}, sid, \mathsf{S})$ to $\mathcal{A}^*$.
- On $(\mathsf{RcvComplete}, sid, \mathsf{P}, \mathsf{S}^*)$ from $\mathcal{A}^*$, retrieve $\langle \mathsf{P}, \mathsf{S}, x \rangle$; abort if there is no such record, or $\mathsf{tickets}(\mathsf{S}^*) = 0$. Else set $\mathsf{tickets}(\mathsf{S}^*)$−− and send $(\mathsf{Eval}, sid, F_{\mathsf{S}^*}(x))$ to P (pick $F_{\mathsf{S}^*}(x) \leftarrow_{\mathrm{R}} \{0,1\}^\ell$ if undefined).

Figure 3.1: Functionality $\mathcal{F}_{\text{OPRF}}$

Description of $\mathcal{F}_{\text{OPRF}}$. The functionality $\mathcal{F}_{\text{OPRF}}$ involves clients, senders and an (ideal-world) adversary, denoted C, S and $\mathcal{A}^*$, respectively. We denote by ℓ the output size of the PRF.

OPRF evaluation is triggered by an $(\mathsf{Eval}, sid, \mathsf{S}, x)$ command from client C requesting the computation of the PRF of server S on input x. Commands $\mathsf{SvrComplete}$ and $\mathsf{RcvComplete}$ denote the completion of S's and C's computation, respectively. Operation $(\mathsf{Eval}, sid, \mathsf{S}, x)$ from a client C with server S is completed by a $(\mathsf{RcvComplete}, sid, \mathsf{C}, \mathsf{S}^*)$ where S^* is the identity of a server that is specified by $\mathcal{A}^*$ and, in the case S is corrupted, it may or may not be equal to the server S specified by C in the Eval command. In the case $\mathsf{S} = \mathsf{S}^*$ we have a computation with the intended server S, while the case $\mathsf{S} \neq \mathsf{S}^*$ corresponds to the attacker channeling the request to a different server, possibly a corrupted one. In fact, we allow the adversary to specify a value S^* which may not even be an identity of any physical server, and each value S^* is interpreted as a pointer to an independent random function $F_{\mathsf{S}^*}(\cdot)$. Thus, there is no guarantee of correctness of the evaluation request and, moreover, two OPRF requests for a same (S, x) pair can be answered differently if S is corrupted. This is where our OPRF formalism $\mathcal{F}_{\text{OPRF}}$ differs fundamentally from the definition of $\mathcal{F}_{\text{VOPRF}}$ in [43] which ensures correct and client-verifiable OPRF computation. This relaxation simplifies the OPRF functionality by dispensing with two essential elements in $\mathcal{F}_{\text{VOPRF}}$, namely, the "public parameters" π associated with each server (and used by the client to check correct evaluation) and the requirement that even corrupted senders must commit to computing an arbitrary but deterministic function (represented in $\mathcal{F}_{\text{VOPRF}}$ by the circuit M).

While we allow $\mathcal{A}^*$ to route a client's request to the wrong server, we do make sure that $\mathcal{A}^*$ cannot forge computations by honest servers. As in [43] this is enforced via a "ticketing mechanism" that ensures that for any honest server S, the number of client-completed OPRF evaluations (i.e., $\mathsf{RcvComplete}$ activations) with S is no more than the number of $\mathsf{SvrComplete}$ activations of S. Specifically, each server S is associated with a ticket value $\mathsf{tickets}(\mathsf{S})$. Each

time that a server S completes its interaction with a client, $\mathsf{tickets}(\mathsf{S})$ increases by 1; each time a client, either honest or corrupted, completes an interaction that is associated to S, $\mathsf{tickets}(\mathsf{S})$ decreases by 1 (provided that it was not 0). This ticketing approach dispenses with the need to extract clients' inputs when building a simulator for proving the security of a given realization of the functionality. This simplification (which is shared with [43]), together with the relaxation of the verifiability property as discussed above, allows for the very simple and efficient OPRF realization presented in the next section.

On the "strong (pseudo)randomness" of functionality $\mathcal{F}_{\text{OPRF}}$. Functionality $\mathcal{F}_{\text{OPRF}}$ keeps record of a family of functions F for the results of the PRF evaluation by different servers (including corrupted ones) on requested inputs. Specifically, $F_{\mathsf{S}}(x)$ is defined as the value of the PRF tied to some server identity S (honest, corrupted, or created by an adversary), on input x specified in some Eval command which is completed by a subsequent $\mathsf{RcvComplete}$ with server tag S (see above on the semantics of the Eval-$\mathsf{RcvComplete}$ sequence). The function values are initially undefined and are picked at random by $\mathcal{F}_{\text{OPRF}}$ upon $\mathsf{RcvComplete}$ activations. We note that $F_{\mathsf{S}}(x)$ is chosen at random even in case that server S is corrupted and/or it corresponds to some virtual server created by the adversary. As a result, any realization of $\mathcal{F}_{\text{OPRF}}$ needs to ensure that OPRF evaluations with corrupted servers result in outputs which are (pseudo)random even to the adversary: Note that $\mathcal{F}_{\text{OPRF}}$ provides the adversary with direct access to all function input-output tables by allowing $\mathcal{A}^*$ to issue Eval requests to $\mathcal{F}_{\text{OPRF}}$. In particular, for the case where S^* is either the identity of a corrupted server or a virtual identity established by $\mathcal{A}^*$, an adversary can evaluate function $F_{\mathsf{S}^*}(\cdot)$ on any argument x *only* by a series of explicit calls to $\mathcal{F}_{\text{OPRF}}$, e.g., $(\mathsf{Eval}, sid, \mathsf{S}^*, x)$, $(\mathsf{SvrComplete}, sid)$ (sent via S^*), and $(\mathsf{RcvComplete}, sid, \mathcal{A}^*, \mathsf{S}^*)$. This means that $\mathcal{F}_{\text{OPRF}}$ learns all inputs on which $\mathcal{A}^*$ evaluates the OPRF functions controlled by $\mathcal{A}^*$, and that these adversarial evaluations must even respect the ticket-counter mechanism. In particular, this implies that if a real-world

adversary $\mathcal{A}$ locally evaluates the (O)PRF on an adversarially chosen key k and any x then (1) the output $F_k(x)$ will appear pseudorandom to $\mathcal{A}$, and (2) an efficient simulator will extract the argument x (although not necessarily the key k) whenever such local evaluation occurs. This strongly suggests that an efficient construction of such $\mathcal{F}_{\text{OPRF}}$ requires ROM (or some other non-black-box model).

It follows that functionality $\mathcal{F}_{\text{OPRF}}$ we introduce is not a strict weakening of the verifiable OPRF functionality $\mathcal{F}_{\text{VOPRF}}$ of [43]: Functionality $\mathcal{F}_{\text{VOPRF}}$ is stronger than $\mathcal{F}_{\text{OPRF}}$ in the sense that it ensures that value y which C receives in response (Eval, sid, y) to request $(\mathsf{Eval}, sid, \mathsf{S}, x)$ satisfies $y = F_{\mathsf{S}}(x)$, while functionality $\mathcal{F}_{\text{OPRF}}$ ensures only that $y = F_{\mathsf{S}^*}(x)$ (in case S is corrupted) for a well-defined but arbitrary S^* specified by $\mathcal{A}^*$. However, in the case of corrupted servers S, functionality $\mathcal{F}_{\text{VOPRF}}$ allows the adversary to specify $F_{\mathsf{S}}(\cdot)$ as an arbitrary circuit, which means that values $F_{\mathsf{S}}(\cdot)$ are predictable to $\mathcal{A}^*$ and that $\mathcal{A}^*$ can evaluate them locally without the $\mathcal{F}_{\text{VOPRF}}$'s knowledge. By contrast, $\mathcal{F}_{\text{OPRF}}$ ensures that $F_{\mathsf{S}}(\cdot)$ is a random function even for corrupted server identities S.

The 2HashDH protocol. In Figure 3.2, we present an efficient realization of $\mathcal{F}_{\text{OPRF}}$ in ROM, based on the 2HashDH-NIZK construction of $\mathcal{F}_{\text{VOPRF}}$ in [43], from which we eliminate the zero-knowledge proofs and the corresponding "public keys" of servers.

This construction relies on a cyclic group $\mathbb{G}$ of prime order m, with g being a generator. The private key k is chosen at random from $\mathbb{Z}_m$. Each client C records tuples of the form (S, x, r, y). The construction uses two hash functions, $H(\cdot,\cdot)$ and $H'(\cdot)$ (hence the name 2HashDH), both modeled as random oracles.

The PRF is defined as $F_k(x) = H(x, H'(x)^k)$. For each value x the client C wants to evaluate in an OPRF instance, C picks a random element r in $\mathbb{Z}_m$, which can be used for OPRF evaluations with the same x but different servers. When C wants to compute $F_k(x)$

Let $H(\cdot,\cdot)$ and $H'(\cdot)$ be hash functions with ranges $\{0,1\}^\ell$ and $\mathbb{G}$, respectively (modeled as random oracles).

- On input $(\mathsf{Eval}, sid, \mathsf{S}, x)$, C proceeds as follows:
  - If there is a record $\langle \mathsf{S}, x, r, y\rangle$, C outputs (Eval, sid, y).
  - Else if there is a record $\langle \mathsf{S}', x, r, y\rangle$ (where $\mathsf{S}' \neq \mathsf{S}$), C records $\langle \mathsf{S}, x, r, \perp\rangle$ and sends $a := H'(x)^r$ to S.
  - Else C picks $r \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, records $\langle \mathsf{S}, x, r, \perp\rangle$ and sends $a := H'(x)^r$ to S.
- On input $(\mathsf{SvrComplete}, sid)$ and a from C, S sends $b := a^k$ to C.
- On b from S, C retrieves $\langle \mathsf{S}, x, r, \perp\rangle$, replaces $\perp$ with $y := H(x, b^{1/r})$ and outputs (Eval, sid, y).

Figure 3.2: Protocol 2HashDH (for PRF output length ℓ)

where k is the private key of a specific server S, it sends $a := H'(x)^r$ to S; S sends back $b := a^k = H'(x)^{rk}$ to C, and C outputs $y := H(x, b^{1/r}) = H(x, H'(x)^k)$.

We describe the protocol in detail in Figure 3.2.

Security analysis. We prove the security of the 2HashDH protocol under the Gap OMDH assumption:

Theorem 1. *Suppose that the $(Q + q_{\mathrm{H}'}, Q)$-Gap OMDH assumption holds for $(\mathbb{G}, g, m)$, where Q is the number of* Eval *messages sent to* C *and $q_{\mathrm{H}'}$ is the number of $H'(\cdot)$ queries. Then protocol 2HashDH in Figure 3.2 realizes functionality $\mathcal{F}_{\mathrm{OPRF}}$ in ROM.*

Proof. Let $N = Q + q_{\mathrm{H}'}$. For any efficient adversary $\mathcal{A}$ against the protocol, we construct a simulator SIM as in Figure 3.3. To keep notation brief we denote functionality $\mathcal{F}_{\mathrm{OPRF}}$ as $\mathcal{F}$.

We now argue that for any efficient environment $\mathcal{Z}$, its view in the simulated ideal world (henceforth simulated world) which SIM produces and its view in the real world are indistinguishable. Without loss of generality, suppose $\mathcal{A}$ is a "dummy" adversary who

1. Pick $r_1, \ldots, r_N \leftarrow_{\mathrm{R}} \mathbb{Z}_m$ and compute $g_1 := g^{r_1}, \ldots, g_N := g^{r_N}$. Set counter $J := 1$.
2. On $\mathcal{A}$ making a fresh query $H'(x)$, answer it with g_J and record (x, r_J, g_J). After that, set $J{+}{+}$.
3. On $(\mathsf{Eval}, sid, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}$, record $\langle \mathsf{C}, \mathsf{S}, r_J, g_J \rangle$ and send g_J to $\mathcal{A}$ as C's message to S. After that, set $J{+}{+}$.
4. On $(\mathsf{SvrComplete}, sid, \mathsf{S})$ from $\mathcal{F}$ and a from $\mathcal{A}$ as some client C's message to S, find record $\langle \mathsf{S}, k, z \rangle$. If there is no such record, pick $k \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, compute $z := g^k$ and record $\langle \mathsf{S}, k, z \rangle$. Regardless, send a^k to $\mathcal{A}$ as S's response to C.
5. On b from $\mathcal{A}$ as some server S's message to a client C, retrieve $\langle \mathsf{C}, \mathsf{S}, r_j, g_j \rangle$ and find record $\langle \mathsf{S}', \cdot, z \rangle$ such that $b = z^{r_j}$. If there is no such record, create a new server identity S' and record $\langle \mathsf{S}', \perp, b^{1/r_j} \rangle$. Regardless, send $(\mathsf{RcvComplete}, sid, \mathsf{C}, \mathsf{S}')$ to $\mathcal{F}$.
6. On $\mathcal{A}$ making a fresh query $H(x, u)$,
   (a) If there is a record (x, r_j, g_j) and a record $\langle \mathsf{S}', \cdot, z \rangle$ such that $u = z^{r_j}$, send $(\mathsf{Eval}, sid, \mathsf{S}', x)$ and then $(\mathsf{RcvComplete}, sid, \mathcal{A}^*, \mathsf{S}')$ to $\mathcal{F}$.
   If $\mathcal{F}$ ignores this message, output fail and abort.
   Else on $\mathcal{F}$'s response (Eval, sid, y), set $H(x, u) := y$.
   (b) Else pick $H(x, u) \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$.

Figure 3.3: The simulator SIM for the 2HashDH protocol

merely passes through all its messages to and from $\mathcal{Z}$, and all its computation to $\mathcal{Z}$.

The argument of indistinguishability uses a sequence of games. We start from $\mathcal{Z}$'s interaction with other parties in the real world, and end at the simulated world. For each two adjacent games $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$, we show that $\mathcal{Z}$'s distinguishing advantage between them is negligible. We denote such distinguishing advantage as $\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_i, \mathbf{G}_{i+1}}$, i.e.,

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_i, \mathbf{G}_{i+1}} = |\Pr[\mathcal{Z} \text{ outputs 1 in } \mathbf{G}_i] - \Pr[\mathcal{Z} \text{ outputs 1 in } \mathbf{G}_{i+1}]|.$$

$\mathbf{G}_0$ is the real world.

In $\mathbf{G}_1$, we make two changes: (1) For every server S, record its key and the corresponding group element; that is, record $\langle \mathsf{S}, k, z := g^k \rangle$. (2) When C computes a or $\mathcal{A}$ queries $H'(x)$,

record the discrete logarithm of $H'(x)$; that is, pick $h \leftarrow_{\mathrm{R}} \mathbb{Z}_m$ and record $(x, h, H'(x) := g^h)$. Obviously,

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_0,\mathbf{G}_1} = 0.$$

In $\mathbf{G}_2$, for an input $(\mathsf{Eval}, sid, \mathsf{S}, x)$ to C, while computing a, pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, set $a := g^w$ and record $\langle \mathsf{C}, \mathsf{S}, w, a \rangle$; also, pick $h \leftarrow_{\mathrm{R}} \mathbb{Z}_m$ and record $(x, h, H'(x) := g^h)$. Then when $\mathcal{A}$ sends b to C, retrieve $(x, h, \cdot)$ and C outputs $(\mathsf{Eval}, sid, y := H(x, b^{h/w}))$.

The difference between $\mathbf{G}_1$ and $\mathbf{G}_2$ is that in $\mathbf{G}_1$, a is defined as $H'(x)^r = g^{hr}$, where $h, r \leftarrow_{\mathrm{R}} \mathbb{Z}_m$; and y is defined as $b^{1/r}$. In $\mathbf{G}_2$, hr is replaced with $w \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, and $1/r$ is replaced with h/w. Clearly this does not change the distribution of a or y, so we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_1,\mathbf{G}_2} = 0.$$

In $\mathbf{G}_3$, on b from $\mathcal{A}$ to C, retrieve $\langle \mathsf{C}, \mathsf{S}, w, a \rangle$ and find record $\langle \mathsf{S}', \cdot, z \rangle$ such that $b = z^w$. If there is no such record, create a new server identity S' and record $\langle \mathsf{S}', \perp, b^{1/w} \rangle$ in addition. (Note that in this way we guarantee that there is always a record $\langle \mathsf{S}', \cdot, z \rangle$ such that $b = z^w$, or equivalently, $z = b^{1/w}$.) Obviously,

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_2,\mathbf{G}_3} = 0.$$

In $\mathbf{G}_4$, C outputs $(\mathsf{Eval}, sid, y \leftarrow_{\mathrm{R}} \{0,1\}^\ell)$ unless $\mathcal{A}$ queries $H(x, u)$, and there is a record (x, h, g^h) and a record $\langle \mathsf{S}', \cdot, z \rangle$ such that $u = g^h$ (mark this case $(*)$). In addition, if $\mathcal{A}$ makes such $H(x, u)$ query after C outputs $(\mathsf{Eval}, sid, y \leftarrow_{\mathrm{R}} \{0,1\}^\ell)$, set $H(x, u) := y$.

In $\mathbf{G}_3$, y is random in $\mathcal{Z}$'s view unless and until $\mathcal{A}$ queries $H(x, u = b^{h/w})$ (where h is the value in the record $(x, h, H'(x))$). Call this a "crucial query (on u)." We have pointed out that there must be a record $\langle \mathsf{S}', \cdot, z \rangle$ such that $z = b^{1/w}$. Therefore, if $\mathcal{A}$ makes a "crucial query," there must be a record (x, h, g^h) and a record $\langle \mathsf{S}', \cdot, z \rangle$ such that $u = z^h$, so case $(*)$

happens, hence $\mathbf{G}_3$ and $\mathbf{G}_4$ are identical. (If $\mathcal{A}$ makes no "crucial query," $\mathbf{G}_3$ and $\mathbf{G}_4$ are also identical). We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_3,\mathbf{G}_4} = 0.$$

Note that in $\mathbf{G}_4$, for every input $(\mathsf{Eval}, sid, \mathsf{S}, x)$ to C, there is always a record $(x, h, H'(x) = g^h)$. In $\mathbf{G}_5$, do not record $(x, h, H'(x))$ and leave $H'(x)$ undefined unless and until $\mathcal{A}$ queries $H'(x)$.

In $\mathbf{G}_4$, h and $H'(x)$ are used only when $\mathcal{A}$ makes a "crucial query" on $g^h = H'(x)$. Therefore, $\mathcal{Z}$'s views in $\mathbf{G}_4$ and $\mathbf{G}_5$ are identical unless $\mathcal{A}$ does not query $H'(x)$ but makes a "crucial query" on $H'(x)$. If $\mathcal{A}$ does not query $H'(x)$, then $H'(x)$ is a random element in $\mathbb{G}$. Assuming that $\mathcal{A}$ queries $H(\cdot,\cdot)$ q_{H} times, for every $H'(x)$, the probability that $\mathcal{A}$ makes a "crucial query" on $H'(x)$ is at most q_{H}/m; therefore, assuming that there are q_{C} client sessions, there are q_{C} $H'(x)$'s, so the probability that $\mathcal{A}$ makes a "crucial query" on any $H'(x)$ is at most $q_{\mathrm{C}}q_{\mathrm{H}}/m$. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_4,\mathbf{G}_5} \leq \frac{q_{\mathrm{C}}q_{\mathrm{H}}}{m},$$

which is a negligible function of κ.

$\mathbf{G}_6$ is the simulated world. We can see that the only difference between $\mathbf{G}_5$ and $\mathbf{G}_6$ is that $\mathbf{G}_6$ aborts when a certain event fail (as in step 6 in the description of SIM) occurs.

Now we upper bound $\Pr[\mathsf{fail}]$. Event fail occurs in case (a) of step 6, where SIM sends a $(\mathsf{RcvComplete}, sid, \mathcal{A}^*, \mathsf{S})$ message to $\mathcal{F}$ but it is ignored. This can only happen when $\mathsf{tickets}(\mathsf{S}) = 0$. For every server $\tilde{\mathsf{S}}$, let $\mathsf{fail}(\tilde{\mathsf{S}})$ be the particular event that the $(\mathsf{RcvComplete}, sid, \mathcal{A}^*, \tilde{\mathsf{S}})$ message is ignored. Then

$$\Pr[\mathsf{fail}] \leq \sum_{\tilde{\mathsf{S}}} \Pr[\mathsf{fail}(\tilde{\mathsf{S}})].$$

We now upper bound $\Pr[\mathsf{fail}(\tilde{\mathsf{S}})]$ by reducing $\mathsf{fail}(\tilde{\mathsf{S}})$ to the (N, Q)-Gap OMDH problem in $(\mathbb{G}, g, m)$. The reduction, $\mathcal{R}_{\tilde{\mathsf{S}}}$, is shown in Figure 3.4.

Observe the following two facts for $\mathcal{R}_{\tilde{\mathsf{S}}}$:

- Every time the $(\cdot)^{\tilde{k}}$ oracle is queried (in step 4), there is a $(\mathsf{SvrComplete}, \mathit{sid}, \tilde{\mathsf{S}})$ message, so $\mathsf{tickets}(\tilde{\mathsf{S}})$ increments.
- Every time $\mathsf{tickets}(\tilde{\mathsf{S}})$ decrements (in the first case of step 5 or the first condition of the first case of step 6), a pair of the form $(g_j, g_j^{\tilde{k}})$ is recorded (as underlined).

Therefore, if $\mathsf{fail}(\tilde{\mathsf{S}})$ occurs, the number of pairs recorded is more than the number of DDH oracle queries by one, so $\mathcal{R}_{\tilde{\mathsf{S}}}$ solves the (N, Q)-Gap OMDH problem in $(\mathbb{G}, g, m)$. Therefore, we have that

$$\Pr[\mathsf{fail}(\tilde{\mathsf{S}})] \leq \mathbf{Adv}^{\text{Gap-OMDH},\mathbb{G}}_{\mathcal{R}_{\tilde{\mathsf{S}}}},$$

so

$$\mathbf{Dist}^{\mathbf{G}_5,\mathbf{G}_6}_{\mathcal{Z}} \leq \Pr[\mathsf{fail}] \leq \sum_{\tilde{\mathsf{S}}} \Pr[\mathsf{fail}(\tilde{\mathsf{S}})] \leq \sum_{\tilde{\mathsf{S}}} \mathbf{Adv}^{\text{Gap-OMDH},\mathbb{G}}_{\mathcal{R}_{\tilde{\mathsf{S}}}},$$

which is a negligible function of κ.

Summing up all results above, we conclude that $\mathcal{Z}$'s distinguishing advantage between the real world and the simulated world is a negligible function of κ. This completes the proof. □

3.3 The Threshold OPRF Functionality $\mathcal{F}_{\text{TOPRF}}$ and Its Realization

In this section, we provide a formal definition of the Threshold Oblivious PRF (T-OPRF) notion, namely as a secure realization of a UC functionality $\mathcal{F}_{\text{TOPRF}}$ shown in Figure 3.5. The

On input $(y = g^{\tilde{k}}, g_1, \ldots, g_N)$ as an instance of the (N, Q)-Gap OMDH problem in $(\mathbb{G}, g, m)$, $\mathcal{R}_{\tilde{\mathsf{S}}}$ runs the code of the simulator SIM, with the following changes:

- In step 1, only set $J := 1$ and omit all other processes.
- In step 2 and step 3, use the challenges instead of random elements in $\mathbb{G}$ as $g_1, \ldots, g_N$. Furthermore, since there is no r_j, only record (x, g_j) (in step 2) and $\langle \mathsf{C}, \mathsf{S}, g_j \rangle$ (in step 3).
- In step 4, if $\tilde{a}$ acts as a message to $\tilde{\mathsf{S}}$, then use the $(\cdot)^{\tilde{k}}$ oracle to compute $\tilde{b} := \tilde{a}^{\tilde{k}}$ and record $(\tilde{a}, \tilde{b})$ instead of $\left\langle \tilde{\mathsf{S}}, \tilde{k}, \tilde{z} := g^{\tilde{k}} \right\rangle$.
- In step 5, do the following instead: On b from $\mathcal{A}$ as some server S''s message to a client C, retrieve $\langle \mathsf{C}, \mathsf{S}, g_j \rangle$ and do:
  - If there is a record $(\tilde{a}, \tilde{b})$ and $\mathsf{DDH}(g_j, b, \tilde{a}, \tilde{b})$, then $\underline{\text{record } (g_j, b)}$ and send $(\mathsf{RcvComplete}, sid, \mathsf{C}, \mathsf{S}', \tilde{\mathsf{S}})$ to $\mathcal{F}$.
  - Else if there is a record $\left\langle \mathsf{S}, k, z = g^k \right\rangle$ such that $b = a^k$, then send $(\mathsf{RcvComplete}, sid, \mathsf{C}, \mathsf{S}', \mathsf{S})$ to $\mathcal{F}$.
  - Else if there is a record $(a_{\mathsf{S}^*}, b_{\mathsf{S}^*})$ such that $\mathsf{DDH}(g_j, b, a_{\mathsf{S}^*}, b_{\mathsf{S}^*})$, then send $(\mathsf{RcvComplete}, sid, \mathsf{C}, \mathsf{S}', \mathsf{S}^*)$ to $\mathcal{F}$.
  - Else create a new server identity S^*, set $a_{\mathsf{S}^*} := g_j$ and $b_{\mathsf{S}^*} := b$, and send $(\mathsf{RcvComplete}, sid, \mathsf{C}, \mathsf{S}', \mathsf{S}^*)$ to $\mathcal{F}$.
- In case (a) of step 6 (i.e., there is a record (x, g_j)), use the following criteria to determine S' (and then proceed as SIM):
  - If $\mathsf{DDH}(g_j, u, \tilde{a}, \tilde{b})$, then $\underline{\text{record } (g_j, u)}$ and set $\mathsf{S}' := \tilde{\mathsf{S}}$.
  - Else if there is a record $\left\langle \mathsf{S}, k, z = g^k \right\rangle$ such that $u = g_i^k$, then set $\mathsf{S}' := \mathsf{S}$.
  - Else if there is a pair $(a_{\mathsf{S}^*}, b_{\mathsf{S}^*})$ such that $\mathsf{DDH}(g_i, u, a_{\mathsf{S}^*}, b_{\mathsf{S}^*})$, then set $\mathsf{S}' := \mathsf{S}^*$.

  Finally, if $\mathsf{fail}(\tilde{\mathsf{S}})$ occurs, output the set of all underlined recorded pairs.

Figure 3.4: The reduction $\mathcal{R}_{\tilde{\mathsf{S}}}$ to the Gap OMDH problem (for a single server $\tilde{\mathsf{S}}$)

$\mathcal{F}_{\text{TOPRF}}$ functionality defined here is a generalization of the single-server OPRF functionality $\mathcal{F}_{\text{OPRF}}$ in Section 3.2 to the multi-party setting. In the $\mathcal{F}_{\text{TOPRF}}$ setting, the PRF key is effectively controlled by a collection of n servers, and it remains secret as long as no more than a threshold t of these servers are corrupted (hence $\mathcal{F}_{\text{OPRF}}$ is a specific case of $\mathcal{F}_{\text{TOPRF}}$ where $(t, n) = (0, 1)$). Then we show 2HashTDH, a protocol which realizes such (t, n)-threshold "collective control" over a functionality using secret-sharing.

3.3.1 The Threshold OPRF Functionality $\mathcal{F}_{\text{TOPRF}}$

The T-OPRF functionality of Figure 3.5 has two phases, initialization and evaluation. The T-OPRF functionality enforces that the values of any such function remain random to the adversary, similarly as the single-server OPRF notion in Section 3.2, even in the case that the adversary controls the private key and/or its sharing among the n servers (but is not privy to the value the T-OPRF is evaluated on).

In more details, in the initialization phase, a set of n servers, denoted $\mathcal{SI}$, are activated at the discretion of the adversary. The phase is complete when all servers become active. Note that the set may include adversarial servers, yet the functionality guarantees that all servers identified in $\mathcal{SI}$ become active by the end of the initialization phase. During this phase a ticket counter associated with the function controlled by the set of servers is initialized, and so is an empty table implementing the random function shared by the $\mathcal{SI}$ servers.

In the evaluation phase, clients connect to an arbitrary set of servers $\mathcal{SE}$ chosen by the adversary and which may arbitrarily overlap with $\mathcal{SI}$ (representing the fact that the client has no memory of who the servers in $\mathcal{SI}$ are). When, at the discretion of the adversary, a server $\mathsf{S} \in \mathcal{SI}$ completes its interaction, the functionality increases the counter $\mathsf{tickets}(p, \mathsf{S})$. Eventually, the adversary can trigger a response to the client which will be the output from one of the functions recorded by the functionality. Recall that in addition to the proper

In the description below, we assume that $\mathsf{P} \in \{\mathsf{C}, \mathcal{A}^*\}$.
Public Parameter: PRF output-length ℓ, polynomial in security parameter κ.
For every server S, initialize $\mathsf{tickets}(\mathsf{S})$ to 0.
Initialization

- On $(\mathsf{Init}, sid, \mathcal{SI})$ from S where $|\mathcal{SI}| = n$, ignore this message if S is marked active. Else mark S as active. Furthermore, if there is no record for $\mathcal{SI}$, pick any previously unused label p, record $\langle \mathcal{SI}, p \rangle$ and send $(\mathsf{Init}, sid, \mathsf{S}, \mathcal{SI}, p)$ to $\mathcal{A}^*$.
- On (Init, sid, p) from $\mathcal{A}^*$, check if p is a label that has not been used before, and if so, record $\langle \mathcal{A}^*, p \rangle$ and send $(\mathsf{Init}, sid, \mathcal{A}^*, p)$ to $\mathcal{A}^*$.
- On $(\mathsf{InitComplete}, sid, \mathsf{S})$ from $\mathcal{A}^*$, retrieve $\langle \mathcal{SI}, p \rangle$; ignore the message if (i) there is no such record, or (ii) $\mathsf{S} \notin \mathcal{SI}$, or (iii) not all servers in $\mathcal{SI}$ are marked active. Else send $(\mathsf{InitComplete}, sid)$ to S and mark S $\mathsf{initialized}$.

Evaluation

- On $(\mathsf{Eval}, sid, ssid, \mathcal{SE}, x)$ from P where $|\mathcal{SE}| = t+1$, retrieve $\langle \mathcal{SI}, p \rangle$ (if $\mathsf{P} = \mathsf{C}$) or $\langle \mathcal{A}^*, p \rangle$ (if $\mathsf{P} = \mathcal{A}^*$); ignore this message if (i) there is no such record, or (ii) there is a record $\langle ssid, \mathsf{P}, \cdot, \cdot, \cdot \rangle$. Else record $\langle ssid, \mathsf{P}, p, \mathcal{SE}, x \rangle$ and send $(\mathsf{Eval}, sid, ssid, \mathsf{P}, \mathcal{SE})$ to $\mathcal{A}^*$.
- On $(\mathsf{SvrComplete}, sid, ssid, \mathsf{S})$ from $\mathcal{A}^*$, retrieve $\langle \mathcal{SI}, p \rangle$; ignore this message if (i) there is no such record, or (ii) $\mathsf{S} \notin \mathcal{SI}$, or (iii) S is not marked $\mathsf{initialized}$. Else set $\mathsf{tickets}(p, \mathsf{S})$++ and send $(\mathsf{SvrComplete}, sid, ssid)$ to S.
- On $(\mathsf{RcvComplete}, sid, ssid, \mathsf{P}, p^*)$ from $\mathcal{A}^*$, retrieve $\langle \mathcal{SI}, p \rangle$ and $\langle ssid, \mathsf{P}, p, \mathcal{SE}, x \rangle$; ignore this message if there is no such record, or $p^* = p$ but $|\{\mathsf{S} \in \mathcal{SI} \mid \mathsf{tickets}(p, \mathsf{S}) > 0\}| \leq t$. Else if $p^* = p$ then set $\mathsf{tickets}(p, \mathsf{S})$−− for any $t+1$ distinct $\mathsf{S} \in \mathcal{SI}$ such that $\mathsf{tickets}(p, \mathsf{S}) > 0$. Finally, send $(\mathsf{Eval}, sid, ssid, F_{p^*}(x))$ to P (pick $F_{p^*}(x) \leftarrow_{\mathrm{R}} \{0,1\}^\ell$ if undefined).

Figure 3.5: Functionality $\mathcal{F}_{\mathrm{TOPRF}}$ with parameters (t, n)

function $F_p(\cdot)$ the adversary can register additional functions $F_{p^*}(\cdot)$ and may connect an evaluation request from a client to any such function of its choice.

The security guarantees provided by the T-OPRF functionality are the following: (1) it enforces the use of the proper function $F_p(\cdot)$ whenever the set of servers $\mathcal{SE}$ selected for an evaluation are all honest; (2) it "charges" $t+1$ server tickets for accessing the proper function $F_p(\cdot)$ by decrementing (non-zero) ticket counters $\mathsf{tickets}(p, \mathsf{S})$ for an arbitrary set of $t+1$ servers in $\mathcal{SI}$; and (3) the outputs of all functions F (the proper function $F_p(\cdot)$ as well as any additional ones set by the adversary with $p^* \neq p$) are picked at random, chosen on demand as the functionality responds back to the client. These guarantees ensure that at least $t - t' + 1$ honest servers from $\mathcal{SI}$ need to be contacted for the proper function to be evaluated once. To see why this is the case, observe that $t+1$ tickets are "spent" (decremented) during evaluation which correspond to at least $t - t' + 1$ tickets from honest ticketing counters. This implies that $t+1$ servers in $\mathcal{SI}$ have registered a $\mathsf{SvrComplete}$ message as this is the only event that triggers a counter increment. In the real world this corresponds to the event that a server has completed its interaction with a client that attempts to perform an evaluation.

It is important to highlight that the functionality does not necessarily decrement the ticketing counters of the servers identified in the chosen evaluation set $\mathcal{SE}$; rather, it decrements an arbitrary set of $t+1$ non-zero counters for servers in $\mathcal{SI}$. This reflects the fact that the functionality does not provide any guarantee about the identities of the responding servers. For instance, this means that we allow for an implementation of T-OPRF where an honest client C attempts to connect to a set of servers $\mathcal{SE}_1$ that are corrupted and its message is rerouted by the adversary so that, unbeknownst to C, an honest set of servers $\mathcal{SE}_2$ becomes the responder set.

Another important point regarding the T-OPRF functionality is that while it guarantees correct OPRF evaluation in case the client completes an undisturbed interaction with $t+1$

honest servers in $\mathcal{SI}$, the ideal adversary may also maintain an arbitrary collection of random function input-output tables and connect a client to them if desired, as long as the responder set is not composed of honest servers only. For instance, the adversary can assign to a subset of corrupted servers connects to. We stress that this does not jeopardize the privacy of the input value x in any way. At the same time, observe that the randomness requirement imposed for adversarial function outputs restricts our ability to implement the functionality to ROM (just as in OPRF).

3.3.2 Generic T-OPRF Construction from Any OPRF

One can use generic Multi-Party Computation (MPC) to convert *any* OPRF protocol into a threshold OPRF protocol. The following is a blueprint for a T-OPRF with parameters (t, n) given an OPRF protocol, a Message Authentication Code (MAC) scheme, and a generic MPC protocol:

1. *Initialization:* The initialization runs a (t, n)-threshold MPC for the C-S initialization protocol of the OPRF, where S's output state k is replaced by the secret-sharing $(k_1, \ldots, k_n)$ of k where each S_i receives k_i. In addition, each pair of servers $(\mathsf{S}_i, \mathsf{S}_j)$ establishes a shared MAC key K_{ij}.

2. *Evaluation:* The client C's evaluation algorithm is as in the underlying OPRF, except that it broadcasts each message to all servers. However, the server's evaluation algorithm is replaced by the following protocol. Let r_i be the randomness S_i chooses in its first protocol message. Then in each protocol round p the servers do the following:

(1) $\mathsf{S}_1, \ldots, \mathsf{S}_n$ agree on the message $a^{(p)}$ which C sent in this protocol round as follows: For every i and j, S_i sends a MAC on this message using key K_{ij} to S_j. S_j aborts if no server sends a valid MAC on $a^{(p)}$ to S_j.

(2) $\mathsf{S}_1, \ldots, \mathsf{S}_n$ run a (t, n)-threshold MPC protocol for computing S's response in p-th round of the OPRF protocol, given the public input C's messages $a^{(1)}, \ldots, a^{(p)}$ and S's local input k and r. The local input of S_i in this MPC is (k_i, r_i) where $(k_1, \ldots, k_n)$ is the secret-sharing of k and $r = r_1 \oplus \ldots \oplus r_n$. The MPC protocol computes S's response in the p-th round of the OPRF protocol, and this output is received by C.

When applying this transformation to the OPRF from Figure 3.2 where the only operation by the server is to raise the value a sent by the client to the power of k, we get a T-OPRF protocol where each server S_i first verifies the MAC's on value a from all other servers and then computes an exponentiation a^{k_i} where $k_1, \ldots, k_n$ is a secret sharing of k. This is the same protocol as 2HashTDH below *except for the added MAC-verification round*. While a round of MAC broadcast would be computationally inexpensive, requiring an extra round of interaction would make this protocol less practical. However, while less efficient than 2HashTDH, the security of such generically constructed T-OPRF can be shown based on the same assumption needed for the base OPRF, namely, Gap OMDH. Note that the PPSS protocol can be obtained from this T-OPRF, at the same cost and under the same assumptions, using the T-OPRF-to-PPSS compiler of Section 3.5.

3.3.3 The 2HashTDH Protocol

Here we present our threshold oblivious PRF protocol, 2HashTDH, that instantiates the $\mathcal{F}_{\text{TOPRF}}$ functionality. Thus, 2HashTDH provides a secure T-OPRF for use in general applications and, in particular, as the basis for our PPSS protocol, TOPPSS. The 2HashTDH protocol is formally defined as a realization of $\mathcal{F}_{\text{TOPRF}}$ in Figure 3.6. In a nutshell, it is a threshold version of the single-server 2HashDH OPRF protocol from Section 3.2. The underlying PRF, $F_k(x) = H(x, H'(x)^k)$, remains unchanged, but the key k is shared using Shamir secret-sharing across n servers, where server S_i stores the key share

Let $H(\cdot,\cdot)$ and $H'(\cdot)$ be hash functions with ranges $\{0,1\}^\ell$ and $\mathbb{G}$, respectively (modeled as random oracles).

Initialization

1. On input $(\mathsf{Init}, sid, \mathcal{SI})$ where $|\mathcal{SI}| = n$, S passes this input to $\mathcal{F}_{\mathrm{DKG}}$.
   On $(\mathsf{InitComplete}, sid, y, k_i)$ from $\mathcal{F}_{\mathrm{DKG}}$, S records $\langle \mathcal{SI}, y, i, k_i \rangle$, marks itself active, and outputs $(\mathsf{InitComplete}, sid)$.

Evaluation

1. On input $(\mathsf{Eval}, sid, ssid, \mathcal{SE}, x)$, C picks $r \leftarrow_{\mathrm{R}} \mathbb{Z}_m$ and sends $a := H'(x)^r$ to all $\mathsf{S} \in \mathcal{SE}$.
2. On a from C, S_i, provided it is marked active, computes $b_i := a^{\lambda_i k_i}$ where λ_i is a Lagrange interpolation coefficient for index i and index set $\mathcal{SE}$, sends b_i to C and outputs $(\mathsf{SvrComplete}, sid, ssid)$.
3. When C receives b_i from all $\mathsf{S}_i \in \mathcal{SE}$, it outputs $(\mathsf{Eval}, sid, ssid, H(x, (\prod_{\mathsf{S}_i \in \mathcal{SE}} b_i)^{1/r}))$.

Figure 3.6: Protocol 2HashTDH in the $\mathcal{F}_{\mathrm{DKG}}$-hybrid model

k_i. The initialization of such secret-sharing can be done via a Distributed Key Generation (DKG) for discrete-log-based systems, e.g., [38], and in Figure 3.6 we assume it is done with a UC functionality $\mathcal{F}_{\mathrm{DKG}}$ which we discuss further below.

For evaluation, given any subset $\mathcal{SE}$ of $t+1$ servers, the client C sends to each of them the *same* message $a := H'(x)^r$ for random r in $\mathbb{Z}_m$, exactly as in 2HashDH. If each server $\mathsf{S}_i \in \mathcal{SE}$ returned $b_i := a^{k_i}$, then C could reconstruct the value a^k using standard Lagrange interpolation in the exponent, i.e., $a^k = \prod_{\mathsf{S}_i \in \mathcal{SE}} b_i^{\lambda_i}$ with the Lagrange coefficients λ_i computed using the indexes of servers in $\mathcal{SE}$. After computing a^k, the value of $F_k(x)$ is computed by C by deblinding a^k exactly as in the case of protocol 2HashDH. Note that this takes a single exponentiation for each server and 2 exponentiations for the client (to compute a and to deblind a^k) plus 1 multi-exponentiation for the client to compute the Lagrange interpolation on the b_i values.

We optimize this function evaluation by having each server S_i compute $b_i := a^{\lambda_i k_i}$, which costs 1 exponentiation and $O(t)$ multiplications and divisions in $\mathbb{Z}_m$ to compute λ_i. (Note that S_i

must know set $\mathcal{SE}$ to compute λ_i.) This way C can compute a^k using only t multiplications instead of a multi-exponentiation, and the total costs are 1 exponentiation for each S_i and 2 exponentiations for C.

Protocol 2HashTDH can be also be seen as a simplification of the protocol resulting from a generic transformation of any OPRF to T-OPRF using MPC in Section 3.3.2. The server in 2HashDH computes a^k on input a, and the MPC protocol for it is exactly the *threshold exponentiation* protocol described above, except that this generic OPRF to T-OPRF transformation must assure that the servers perform the MPC sub-protocol on the same input a, and this involves an additional round of server-to-server interaction, which the 2HashTDH protocol avoids.

Our 2HashTDH protocol realizes the UC T-OPRF functionality $\mathcal{F}_{\text{TOPRF}}$ under the T-OMDH assumption in ROM. As we will argue in Section 3.3.4, this implies security under OMDH in ROM in several cases, including the *full corruption* case, where the adversary corrupts $t' = t$ servers, and the *additive sharing* case, where $t = n - 1$. Functionality $\mathcal{F}_{\text{DKG}}$ has well-known efficient realizations in ROM under the Diffie-Hellman assumption which is implied by OMDH, and hence also by T-OMDH.

Note on Distributed Key Generation. Protocol 2HashTDH assumes that servers in $\mathcal{SI}$ establish a secret-sharing $(k_1, \ldots, k_n)$ of a random key k over authenticated channels via a DKG functionality $\mathcal{F}_{\text{DKG}}$, shown in Figure 3.7. The DKG sub-protocol for discrete-log based cryptosystems can be efficiently realized without client's involvement [38, 74], but if the call to initialize a T-OPRF instance is executed by an *honest* client, then the DKG sub-protocol can be even simpler, because the client can generate sharing $(k_1, \ldots, k_n)$ of k and then distribute the shares among the servers in $\mathcal{SI}$. Note that since our realizations of $\mathcal{F}_{\text{TOPRF}}$ pertains only to the *static* adversarial model, where the identity of corrupt parties is determined at the outset, we would not explicitly require that the parties erase the information used in

- On $(\mathsf{Init}, sid, \mathcal{SI})$ from S where $|\mathcal{SI}| = n$, ignore this message if S is marked active or $\mathsf{S} \notin \mathcal{SI}$. Else let $\mathsf{Corrupted}$ be the subset of $\mathcal{SI}$ that is corrupted and $t' := |\mathsf{Corrupted}|$. If there is no record for sid, then if $t' \leq t$ then pick $a_0, a_1, \ldots, a_{t-t'} \leftarrow_{\mathrm{R}} \mathbb{Z}_m$ and record $\langle \mathcal{SI}, a_0, a_1, \ldots, a_{t-t'} \rangle$; else record $\mathcal{SI}$. Regardless, mark S as active and send $(\mathsf{Init}, sid, \mathsf{S}, \mathcal{SI})$ to $\mathcal{A}^*$.
- On $(\mathsf{Init}, sid, \mathcal{SI}, s)$ from a corrupted $\mathsf{S} \in \mathcal{SI}$, if there is a record for $\mathcal{SI}$, record $\langle \mathcal{A}^*, \mathsf{S}, s \rangle$, mark S active and send $(\mathsf{Init}, sid, \mathcal{A}^*, \mathsf{S})$ to $\mathcal{A}^*$.
- On $(\mathsf{InitComplete}, sid, \mathsf{S}_i)$ from $\mathcal{A}^*$, retrieve $\langle sid, \mathcal{SI}, a_0, a_1, \ldots, a_{t-t'} \rangle$; ignore the message if (i) there is no such record, or (ii) $\mathsf{S}_i \notin \mathcal{SI}$, or (iii) not all servers in $\mathcal{SI}$ are marked active. Else send $(\mathsf{InitComplete}, sid, g^{a_0}, i, s_i)$ to S_i and $(\mathsf{InitComplete}, sid, \mathsf{S}_i, g^{a_0})$ to $\mathcal{A}^*$, where $s_i = p(i)$ and $p(x)$ is a polynomial whose first $t - t' + 1$ coefficients match $a_0, a_1, \ldots, a_{t-t'}$ and $p(j) = s_j$ for each j such that $\mathsf{S}_j \in \mathsf{Corrupted}$ and there is a record $\langle \mathcal{A}^*, \mathsf{S}_j, s_j \rangle$.

Figure 3.7: Distributed key generation functionality $\mathcal{F}_{\mathrm{DKG}}$ [74]

initialization, but any implementation should erase such information. In our specification of protocol 2HashTDH we rely on the $\mathcal{F}_{\mathrm{DKG}}$ functionality to abstract from any specific DKG implementation, e.g., whether it is done by the server or by an honest client.

3.3.4 The (Gap) Threshold OMDH Assumption

Additional notations. If $|\vec{a}| = n$ and J is a sequence in $\{1, \ldots, n\}$, then $\vec{a}_J$ denotes the components of $\vec{a}$ with indices in J, i.e., $\vec{a}_J = [a_{i_1}, \ldots, a_{i_k}]^T$ if $J = (i_1, \ldots, i_k)$.

Let $\mathcal{I}_w$ be the set of w-element subsets of $\{1, \ldots, n\}$, i.e., $\mathcal{I}_w = \{I \subseteq \{1, \ldots, n\} \mid |I| = w\}$. Let $W(\vec{a})$ be the Hamming weight of $\vec{a}$. Let $\mathcal{V}_w$ be the set of n-bit binary vectors $\vec{q}$ such that $W(\vec{q}) = w$, i.e., $\mathcal{V}_w = \{\vec{v} \in \{0,1\}^n \mid v_i = 1 \text{ iff } i \in \mathcal{I}_w\}$. For $\vec{q} = [q_1, \ldots, q_n]^T$ define $C_w(\vec{q})$ as the maximum integer m for which there exist $\vec{v}_1, \ldots, \vec{v}_m \in \mathcal{V}_w$ (not necessarily distinct) such that $\vec{v}_1 + \ldots + \vec{v}_m \leq \vec{q}$. In other words, $C_w(\vec{q})$ is the maximum number of times one can subtract elements in $\mathcal{V}_w$ from $\vec{q}$ such that the result remains $\geq \vec{0}$. For example, if $\vec{q} = [3, 3, 4]^T$ then $C_2(\vec{q}) = 5$ because $\vec{q} = 2 \times [1, 0, 1]^T + [1, 1, 0]^T + 2 \times [0, 1, 1]^T$.

T-OMDH intuition. Let $\mathbb{G}$ be a cyclic group of prime order $m > n$. The T-OMDH assumption considers the setting where a random exponent $k \in \mathbb{Z}_m$ is secret-shared using a random t-degree polynomial $p(\cdot)$, and the n trustees holding shares $k_1 = p(1), \ldots, k_n = p(n)$ implement a "threshold exponentiation" protocol which computes a^k for any given $a \in \mathbb{G}$ and $k = p(0)$. Let $\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)$ be an oracle which on input $(i, a) \in \{1, \ldots, n\} \times \mathbb{G}$ outputs $a^{p(i)}$. The standard way to implement threshold exponentiation is to choose a set $I \in \mathcal{I}_{t+1}$, compute $b_i = \mathsf{T\text{-}OMDH}_p(i, a) = a^{k_i}$ for each i in I and derive a^k as $\prod_{i \in I} b_i^{\lambda_i}$ using Lagrange interpolation coefficients λ_i such that $k = \sum_{i \in I} \lambda_i k_i$. The T-OMDH assumption states that querying oracle $\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)$ on at least $t + 1$ different points $i \in \{1, \ldots, n\}$ is *necessary* to compute $a^{p(0)}$ for a given random challenge a. More generally, T-OMDH considers an experiment where the attacker $\mathcal{A}$ receives a challenge set $S = \{g_1, \ldots, g_N\}$ of random elements in $\mathbb{G}$ and is given access to the $\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)$ oracle for random t-degree polynomial p. T-OMDH assumption states that $\mathcal{A}$ can compute g_j^k for $k = p(0)$ for no more than $C_{t+1}(q_1, \ldots, q_n)$ elements $g_j \in S$, where q_i is the number of $\mathcal{A}$'s queries to $\mathsf{T\text{-}OMDH}_p(i,\cdot)$.

The above intuition and Definition 1 below correspond to the setting where the attacker does not control any of the trustees holding shares of p, hence it needs $t + 1$ queries to $\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)$ to compute $a^{p(0)}$ for each random challenge a. Later we extend this definition to the case where $\mathcal{A}$ controls a subset of trustees.

Definition 1. *The (t, n, N, Q)-Threshold One-More Diffie Hellman (T-OMDH) assumption holds in group $\mathbb{G}$ of prime order m if the probability of any polynomial-time adversary $\mathcal{A}$ winning the following game is negligible:*

$\mathcal{A}$, on input $S = \{g_1, \ldots, g_N\}$ where $g_i \leftarrow_{\mathrm{R}} \mathbb{G}$ for $i \in \{1, \ldots, N\}$, is given access to an oracle $\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)$ for a random t-degree polynomial $p(\cdot)$ over $\mathbb{Z}_m$. $\mathcal{A}$ wins if it outputs g_j^k where $k = p(0)$ for $Q + 1$ different elements $g_j \in S$, and if $C_{t+1}(q_1, \ldots, q_n) \leq Q$ where q_i is the number of $\mathcal{A}$'s queries to $\mathsf{T\text{-}OMDH}_p(i,\cdot)$.

Note that the (N, Q)-OMDH assumption [9, 47] is the (t, n, N, Q)-T-OMDH assumption for $t = 0$ and any $n \geq 1$, because then $p(\cdot)$ is a constant polynomial and $C_1(\vec{q}) = W(\vec{q})$, i.e., the total number of $\mathcal{A}$'s $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$ queries.

T-OMDH: the general case. In its general form, the T-OMDH assumption corresponds to computing g_j^k if some subset of $t' \leq t$ trustees holding shares $k_i = p(i)$ is corrupt, and hence the adversary can not only learn these shares but can also set them at will.

Definition 2. *The (t', t, n, N, Q)-T-OMDH assumption holds in group $\mathbb{G}$ of prime order m if for any $\mathsf{Bad} \subseteq \{1, \ldots, n\}$ such that $|\mathsf{Bad}| = t' \leq t$, the probability of any polynomial-time adversary $\mathcal{A}$ winning the following game is negligible:*

$\mathcal{A}$, on input $S = \{g_1, \ldots, g_N\}$ where $g_i \leftarrow_{\mathrm{R}} \mathbb{G}$ for $i \in \{1, \ldots, N\}$, specifies a set of t' values $\{\alpha_j\}_{j \in \mathsf{Bad}}$ in $\mathbb{Z}_m$. A random t-degree polynomial $p(\cdot)$ over $\mathbb{Z}_m$ is then chosen subject to the constraint that $p(j) = \alpha_j$ for $j \in \mathsf{Bad}$, and the adversary $\mathcal{A}$ is given access to oracle $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$. $\mathcal{A}$ wins if it outputs g_j^k where $k = p(0)$ for $Q+1$ different elements $g_j \in S$, and if $C_{t-t'+1}(q_1, \ldots, q_n) \leq Q$ where q_i for $i \notin \mathsf{Bad}$ is the number of $\mathcal{A}$'s queries to $\mathsf{T\text{-}OMDH}_p(i, \cdot)$, and $q_i = 0$ for $i \in \mathsf{Bad}$.

Note that (t', t, n, N, Q)-T-OMDH is identical to (t, n, N, Q)-T-OMDH for $t' = 0$.

Gap T-OMDH. In order to prove the security of T-OPRF, we need to extend the T-OMDH assumption stated in Definition 2 to its "gap" form, i.e., suppose $\mathbb{G}$ is a gap group where $\mathcal{A}$ is in addition given access to the DDH oracle in $\mathbb{G}$.

Definition 3. *The (t', t, n, N, Q)-Gap T-OMDH assumption is the T-OMDH assumption of Definition 2, except that $\mathcal{A}$ is also given access to the $\mathsf{DDH}(\cdot, \cdot, \cdot, \cdot)$ oracle in group $\mathbb{G}$, which on input $(a, b, c, d) \in \mathbb{G}^4$ outputs 1 if $\log_a b = \log_c d$ and 0 otherwise.*

In Theorem 12 in Appendix A we show that the (t, t', n, N, Q)-(Gap)T-OMDH assumption holds in the generic group model for any (t', t, n). Specifically, the advantage of a T-OMDH adversary restricted to r generic group operations is upper-bounded by $O(Qr^2/m)$, assuming $r \geq Q \geq N$. This is larger by factor Q from the $O(r^2/m)$ upper-bounds on generic group attacks against many static problems related to discrete logarithm [71], and this weakening is caused by the presence of up to Q-degree polynomials of the "target" secret $k = p(0)$ in the representation of the group elements which the adversary can compute given access to $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$ using the query pattern $\vec{q} = [q_1, \ldots, q_n]^T$ such that $C_{t-t'+1}(\vec{q}) \leq Q$. Since (N, Q)-OMDH is equivalent to (t', t, n, N, Q)-T-OMDH for $(t', t) = (0, 0)$ and any n, the same upper-bound applies to OMDH, and to the best of our knowledge this is the first generic model security hardness argument for the OMDH (or Gap OMDH) assumption.

T-OMDH = OMDH in full corruption and additive sharing cases. The T-OMDH and OMDH assumptions are equivalent in two important cases, namely the *full corruption* case of $t' = t$, for any (t, n), and in the *additive sharing* case of $t = n - 1$, for any t'. The following two theorems relate the non-gap versions of T-OMDH and OMDH, but equivalent statements hold for the gap versions of these assumptions as well.

Theorem 2. *The (t', t, n, N, Q)-T-OMDH assumption is equivalent to the (N, Q)-OMDH assumption for $t' = t$.*

Proof. If $t' = t$, then the bound $C_{t-t'+1}(\vec{q})$ on Q simplifies to $\sum_{i \notin \mathsf{Bad}} q_i$, i.e., the bound on the number of g_j's for which $\mathcal{A}$ can compute $g_j^{p(0)}$ is the total number of $\mathcal{A}$'s queries to non-corrupted trustees.

Let $\mathcal{A}$ be an adversary against the (t, t, n, N, Q)-T-OMDH assumption making a total number of $Q = \sum_{i \notin \mathsf{Bad}} q_i$ queries, for some t-element set $\mathsf{Bad} = \{\alpha_1, \ldots, \alpha_t\}$ and an assignment $F : \mathsf{Bad} \to \mathbb{Z}_m$ of shares of corrupt trustees. Note that k, Bad and F define a unique

t-degree polynomial $p(\cdot)$ such that $p(0) = k$ and $p(\alpha) = F(\alpha)$ for all $\alpha \in \mathsf{Bad}$. For any $i \in \{1, \ldots, n\} \setminus \mathsf{Bad}$, let $\lambda_{i,0}, \ldots, \lambda_{i,t}$ be the Lagrange coefficients such that $p(i) = \lambda_{i,0} p(0) + \sum_{j=1}^{t} \lambda_{i,j} p(\alpha_j)$. We construct a reduction $\mathcal{R}$ which solves the (N, Q)-OMDH problem as follows:

$\mathcal{R}$, on input $S = \{g_1, \ldots, g_N\}$, passes it to $\mathcal{A}$. On $F : \mathsf{Bad} \to \mathbb{Z}_m$ and $\mathcal{A}$ querying $\mathsf{T\text{-}OMDH}_p(i, a)$, $\mathcal{R}$ queries a^k to compute $b = a^k$, and returns $b' = b^{\lambda_{i,0}} \cdot a^{\lambda_{i,1} F(\alpha_1) + \ldots + \lambda_{i,t} F(\alpha_t)}$ to $\mathcal{A}$. Finally, $\mathcal{R}$ copies $\mathcal{A}$'s output.

We can see that $\mathcal{R}$ consistently answers $\mathcal{A}$'s queries to $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$ for the unique t-degree polynomial $p(\cdot)$ such that $p(0) = k$ and $p(\alpha) = F(\alpha)$ for $\alpha \in \mathsf{Bad}$. Hence in particular if $\mathcal{A}$ wins, i.e., its output includes $g_j^{p(0)}$ for at least $Q + 1$ of g_j's, then $\mathcal{R}$ solves the (N, Q)-OMDH problem on S. We have that

$$\mathbf{Adv}_{\mathcal{R}}^{\mathrm{OMDH},\mathbb{G}} \geq \mathbf{Adv}_{\mathcal{A}}^{\mathrm{T-OMDH},\mathbb{G}},$$

so

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{T-OMDH},\mathbb{G}} \leq \mathbf{Adv}_{\mathcal{R}}^{\mathrm{OMDH},\mathbb{G}},$$

which is a negligible function of κ. □

Theorem 3. *The (t', t, n, N, Q)-T-OMDH assumption is equivalent to the (N, Q)-OMDH assumption for $n = t + 1$.*

Proof. If $n = t + 1$, then shares $k_i = p(i)$ for $i \in \{1, \ldots, n\} \setminus \mathsf{Bad}$ are uniformly random in $\mathbb{Z}_m$, and $p(0) = \sum_{i=1}^{n} \lambda_i p(i)$ for known constants λ_i. Note also that $C_{n-t'}(\vec{q}) = \min_{i \notin \mathsf{Bad}} q_i$, i.e., the bound on the number of g_j's for which $\mathcal{A}$ can compute $g_j^{p(0)}$ is the minimal number of queries the adversary makes to an uncorrupted trustee.

Let $\mathcal{A}$ be an adversary against the (t', t, n, N, Q)-T-OMDH assumption for $n = t + 1$, making q_i queries to the uncorrupted trustee $i \notin \mathsf{Bad}$ such that $\min_{i \notin \mathsf{Bad}} q_i \leq Q$, for some t'-element

subset $\mathsf{Bad} \subseteq \{1, \ldots, n\}$ and an assignment $F : \mathsf{Bad} \to \mathbb{Z}_m$ of shares of corrupt trustees. We construct a reduction $\mathcal{R}$ which solves the (N, Q)-OMDH problem using $\mathcal{A}$ as follows:

$\mathcal{R}$, on input $S = \{g_1, \ldots, g_N\}$, passes it to $\mathcal{A}$. $\mathcal{R}$ picks $i^* \leftarrow_{\mathrm{R}} (\{1, \ldots, n\} \setminus \mathsf{Bad})$ (a guess of the index of the trustee whom $\mathcal{A}$ will query the least), picks shares $k_i \leftarrow_{\mathrm{R}} \mathbb{Z}_m$ for $i \in \{1, \ldots, n\} \setminus (\mathsf{Bad} \cup \{i^*\})$ and sets $k_i := F(i)$ for $i \in \mathsf{Bad}$. On $\mathcal{A}$ querying $\mathsf{T\text{-}OMDH}_p(i, a)$, if $i \neq i^*$, $\mathcal{R}$ returns a^{k_i} to $\mathcal{A}$; if $i = i^*$, $\mathcal{R}$ queries a^k to compute $b = a^k$, and returns $(b \cdot a^{\sum_{i \neq i^*} \lambda_i k_i})^{1/\lambda_i}$ to $\mathcal{A}$. Finally, $\mathcal{R}$ copies $\mathcal{A}$'s output.

We can see that $\mathcal{R}$ consistently answers $\mathcal{A}$'s queries to $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$ for the random $(n-1)$-degree polynomial p such that $p(0) = k$ and $p(i) = F(i)$ for $i \in \mathsf{Bad}$. If $\mathcal{R}$'s guess i^* is correct, then it makes at most $\min_i q_i = Q$ queries to $(\cdot)^k$, and if $\mathcal{A}$ wins, i.e., it computes $g_j^{p(0)} = g_j^k$ for at least $Q + 1$ of g_j's, then $\mathcal{R}$ solves the (N, Q)-OMDH problem on S. We have that

$$\mathbf{Adv}_{\mathcal{R}}^{\mathrm{OMDH},\mathbb{G}} \geq \frac{1}{n} \cdot \mathbf{Adv}_{\mathcal{A}}^{\mathrm{T-OMDH},\mathbb{G}},$$

so

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{T-OMDH},\mathbb{G}} \leq n \cdot \mathbf{Adv}_{\mathcal{R}}^{\mathrm{OMDH},\mathbb{G}},$$

which is a negligible function of κ. □

T-OMDH vs. OMDH for General Threshold Parameters. It is less clear how to relate the T-OMDH and OMDH problems for any t' and t, in the case that $t' < t$, and $t < n - 1$. Consider a specific case that $t' = 0$ and $n = 2(t + 1)$. Adversary $\mathcal{A}$ breaks the T-OMDH assumption if, for example, it computes g_j^k on $2s+1$ challenges after making s queries to $\mathsf{T\text{-}OMDH}_p(i, \cdot)$ for each $i \in \{1, \ldots, n\}$ where $k = p(0)$ and $k_i = p(i)$ for $i \in \{1, \ldots, n\}$. (Note that if $\vec{q} = [s, \ldots, s]^T$ and $n = 2(t + 1)$ then $C_{t+1}(\vec{q}) = 2s$, hence computing g_j^k on $2s + 1$ challenges breaks the assumption.) It is not clear how an efficient reduction $\mathcal{R}$ can break the OMDH problem given access to $\mathcal{A}$, because $\mathcal{R}$ would seemingly have to satisfy the

following constraints: (1) $\mathcal{R}$ would have to make only $2s$ queries to $(\cdot)^k$, but it would have to service s queries to $\mathsf{T\text{-}OMDH}_p(i,\cdot)$ for each i, i.e., ns queries to $\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)$ in total, and $n = 2(t+1) \geq 4$; (2) $\mathcal{R}$ would presumably need to equate secret k in its OMDH challenge with value $p(0)$ in the T-OMDH challenge; (3) $\mathcal{R}$ would presumably need to answer each $\mathsf{T\text{-}OMDH}_p(i,\cdot)$ query consistently, i.e., $\mathcal{R}$ has to reply to $\mathsf{T\text{-}OMDH}_p(i,a)$ with a^{k_i} for some fixed vector of exponents $\vec{k} = [k_1,\ldots,k_n]^T$, because otherwise $\mathcal{A}$ can distinguish interaction with $\mathcal{R}$ from the real security game by checking if $\mathsf{T\text{-}OMDH}_p(i,a)^r = \mathsf{T\text{-}OMDH}_p(i,a^r)$. Since a^r and a are independent group elements for $r \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, it is not clear how $\mathcal{R}$ could detect $\mathcal{A}$'s queries which are designed to test if $\mathcal{R}$ responds to $\mathcal{A}$'s $\mathsf{T\text{-}OMDH}_p(i,\cdot)$ queries with consistent answers; (4) Finally, values $(k_1,\ldots,k_n) = (p(1),\ldots,p(n))$ would need to satisfy linear constraints imposed by the polynomial of degree $t < n-1$, because $\mathcal{A}$ could test that $\mathcal{R}$'s responses to $\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)$ queries satisfy these constraints, similarly as described above. Conditions (2)-(4) can be met e.g., if $\mathcal{R}$ picks $k_i = p(i)$ at random for $i = 1,\ldots,t$, and sets $p(i)$ for $i > t$ as a linear function of $k = p(0)$ and these first t values of $p(\cdot)$. But then it is not clear how $\mathcal{R}$ could reply to any $\mathsf{T\text{-}OMDH}_p(i,\cdot)$ query for $i > t$ without querying $(\cdot)^k$, thus making $(n-t)s = (t+2)s > 2s$ queries to $(\cdot)^k$, violating condition (1).

3.3.5 Security Analysis of 2HashTDH

Theorem 4 below states that protocol 2HashTDH of Figure 3.6 is secure under the T-OMDH assumption in ROM. According to Theorem 2 from Section 3.3.4, Theorem 4 also implies that 2HashTDH is secure under the OMDH assumption in ROM in the full corruption case of $t' = t$. Also, note that Theorem 1 is equivalent to Theorem 4 in the special case of $(t',t,n) = (0,0,1)$.

Theorem 4. *Suppose that the $(t',t,n,Q+q_{\mathrm{H}'},Q)$-Gap T-OMDH assumption holds for $(\mathbb{G},g,m)$, where Q is the number of* Eval *messages sent to* C*, $q_{\mathrm{H}'}$ is the number of $H'(\cdot)$ queries and $t' < t$ is the number of corrupted servers in $\mathcal{SI}$. Then protocol 2HashTDH in*

Figure 3.6 realizes functionality $\mathcal{F}_{\text{TOPRF}}$ with parameters (t, n) in the $\mathcal{F}_{\text{DKG}}$-hybrid world in ROM.

Proof. We only provide a proof sketch. Let $N = Q + q_{\text{H}'}$. The simulator SIM is shown in Figure 3.8. To keep notation brief we denote functionality $\mathcal{F}_{\text{TOPRF}}$ as $\mathcal{F}$.

Similar to the argument for Theorem 1, we conclude that if fail does not occur, $\mathcal{Z}$'s view in the real world and the simulated world are indistinguishable. Now we upper-bound $\Pr[\mathsf{fail}]$. For every *sid*, we reduce $\Pr[\mathsf{fail}]$ for this *sid* to the (t', t, n, N, Q)-Gap T-OMDH problem in $(\mathbb{G}, g, m)$. The reduction, $\mathcal{R}$, is shown in Figure 3.9. It follows that we can bound $\Pr[\mathsf{fail}]$ using q_{T} hybrids, where q_{T} denotes the number of *sid*'s in total.

The reduction $\mathcal{R}$, given $(Q, g_1, \ldots, g_N)$, simulates the UC execution, until the moment that it produces the special private output $(\vec{F}, \sigma)$. Observe that at this moment no $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$ queries have been issued. $\mathcal{R}$ appends to σ the random coins of $\mathcal{Z}$ up until this moment and produces $(\vec{F}, \sigma)$ as the output. In the second stage $\mathcal{R}$ receives σ as the input and is thus capable of continuing the UC execution while enjoying now access to the $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$ with a polynomial $p(\cdot)$ which is suitably defined based on the corrupted servers' values. In this way the oracle queries of SIM can be served using the access that $\mathcal{R}$ has to the $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$ and $\mathsf{DDH}(\cdot, \cdot, \cdot, \cdot)$ oracles. Finally, $\mathcal{R}$ outputs $(J, \boldsymbol{V})$ so that for each pair (g_d, b) recorded in the special output tape of SIM, J contains d and b is included in $\boldsymbol{V}$ (in case no pairs are found $\mathcal{R}$ fails).

We finally argue that $\mathcal{R}'$ will break the T-OMDH assumption with the same probability of success as the fail event. Recall that Event fail occurs in the first case of step 9, when SIM sends a $(\mathsf{RcvComplete}, sid, ssid, \mathcal{A}^*, p)$ message to $\mathcal{F}$ but it is ignored. This can happen only if $|\{\mathsf{S} \mid \mathsf{tickets}(p, \mathsf{S}) > 0\}| \leq t$, i.e., there are not enough servers with positive ticket counters. We can see the following two facts:

1. Pick $r_1, \ldots, r_N \leftarrow_{\mathrm{R}} \mathbb{Z}_m$ and compute $g_1 := g^{r_1}, \ldots, g_N := g^{r_N}$. Set counter $D := 1$.

2. On $\mathcal{A}$ making a fresh query $H'(x)$, answer it with g_D and record (x, r_D, g_D). After that, set $D{+}{+}$.

3. On $(\mathsf{Init}, sid, \mathsf{S}, \mathcal{SI}, p)$ from $\mathcal{F}$, if there is no record for $\mathcal{SI}$, pick $k \leftarrow_{\mathrm{R}} \mathcal{K}$, record $\langle \mathsf{S}, \mathcal{SI}, p, k \rangle$, compute $z := g^k$ and record (p, z). Regardless, mark S active and send $(\mathsf{Init}, sid, \mathsf{S}, \mathcal{SI})$ to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathrm{DKG}}$.

4. On $(\mathsf{Init}, sid, \mathcal{SI}, s)$ from a corrupted $\mathsf{S}^* \in \mathcal{SI}$ aimed at $\mathcal{F}_{\mathrm{DKG}}$, if there is a record for $\mathcal{SI}$, record $\langle \mathcal{A}, \mathsf{S}^*, s \rangle$, mark S^* active and send $(\mathsf{Init}, sid, \mathcal{A}, \mathsf{S}^*)$ to $\mathcal{A}$.

5. On $(\mathsf{InitComplete}, sid, \mathsf{S})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{DKG}}$, retrieve $\langle \mathsf{S}, \mathcal{SI}, p, k, z \rangle$; ignore this message if (i) there is no such record, or (ii) $\mathsf{S} \notin \mathcal{SI}$, or (iii) not all servers in $\mathcal{SI}$ are marked active. Else if there is no record $\langle \mathcal{SI}, k_1, \ldots, k_n \rangle$, create it by setting $k_i := p(i)$ for each $\mathsf{S}_i \in \mathcal{SI}$ where $p(\cdot)$ is a random polynomial subject to the restriction $p(i) = s_i$ for each record of the form $\langle \mathcal{A}, \mathsf{S}_i, s_i \rangle$. Regardless, record $\langle \mathsf{S}_i, \mathcal{SI}, i, k_i \rangle$ and send $(\mathsf{InitComplete}, sid, \mathsf{S})$ to $\mathcal{F}_{\mathrm{TOPRF}}$.

6. On $(\mathsf{Eval}, sid, ssid, \mathsf{C}, \mathcal{SE})$ from $\mathcal{F}$, record $\langle ssid, \mathsf{C}, \mathcal{SE}, r_D, g_D \rangle$ and send $(\mathcal{SE}, g_D)$ as a message to each $\mathsf{S} \in \mathcal{SE}$. After that, set $D{+}{+}$.

7. On $(\mathsf{SvrComplete}, sid, ssid, \mathsf{S}_i)$ from $\mathcal{F}$ and $(\mathcal{SE}, a)$ from $\mathcal{A}$ as some client C's message to server S_i, retrieve $\langle \mathsf{S}_i, \mathcal{SI}, i, k_i \rangle$ (ignore this message if there is no such record or if $\mathcal{SE} \not\subseteq \mathcal{SI}$), compute interpolation coefficient λ_i corresponding to index i and set $\mathcal{SE}$ and send $a^{\lambda_i k_i}$ to $\mathcal{A}$ as S_i's response to C.

8. As soon as b_i for all $\mathsf{S} \in \mathcal{SE}$ defined in a record $\langle ssid, \mathsf{C}, \mathcal{SE}, r_d, g_d \rangle$ have been received from $\mathcal{A}$, compute $b := \prod_{\mathsf{S} \in \mathcal{SE}} b_i$ and find record (p, z) such that $b = z^{r_d}$. If there is no such record, choose a unique label p, record $(p, b^{1/r_d})$ and send $(\mathsf{Init}, sid, \mathcal{A}^*, p)$ to $\mathcal{F}_{\mathrm{TOPRF}}$. Regardless, send $(\mathsf{RcvComplete}, sid, ssid, \mathsf{C}, p)$ to $\mathcal{F}_{\mathrm{TOPRF}}$.

9. On $\mathcal{A}$ making a fresh query $H(x, u)$,

   (a) If there is a record (x, r_d, g_d), find record (p, z) such that $u = z^{r_d}$. If there is no such record, choose a unique label p, record $(p, u^{1/r_d})$ and send $(\mathsf{Init}, sid, \mathcal{A}^*, p)$ to $\mathcal{F}$. Regardless, choose a new unique label $ssid$, send $(\mathsf{SvrComplete}, sid, ssid, \mathsf{S}^*)$ for the t' corrupted servers S^*, $(\mathsf{Eval}, sid, ssid, (\mathcal{A}^*)^{t+1}, x)$ and then $(\mathsf{RcvComplete}, sid, ssid, \mathcal{A}^*, p)$ to $\mathcal{F}$.
   If $\mathcal{F}$ ignores this message, output fail and abort.
   Else on $\mathcal{F}$'s response (Eval, sid, y), set $H(x, u) := y$.

   (b) Else pick $H(x, u) \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$.

Figure 3.8: The simulator SIM for the 2HashTDH protocol

On input $(Q, g_1, \ldots, g_N)$ as an instance of the (t', t, n, N, Q)-Gap T-OMDH problem in $(\mathbb{G}, g, m)$, $\mathcal{R}$ runs the code of the simulator SIM, with the following changes:

- In step 1, only set $D := 1$ and omit all other processes.
- In step 2 and step 6, use the challenges instead of random elements in $\mathbb{G}$ as $g_1, \ldots, g_N$. Furthermore, since there is no r_d, only record (x, g_d) in step 2 and $\langle ssid, \mathsf{C}, \mathcal{SE}, g_d \rangle$ in step 6.
- In step 3, omit the choice of k and only record $\langle \mathsf{S}, \mathcal{SI}, p \rangle$. Furthermore, for all honest server S_i, set $q_i := 0$.
- In step 4, on $(\mathsf{Init}, sid, \mathcal{SI}, s)$ from server S_i which is marked active, add s to the i-th location of the $\vec{F}$ vector. When $\vec{F}$ has all corrupted, t' in number, locations complete, set σ to the complete view of SIM so far and generate $(\vec{F}, \sigma)$ in a special (private) output tape.
- In step 5, in the case that there is no record $\langle \mathcal{SI}, k_1, \ldots, k_n \rangle$, query $\mathsf{T\text{-}OMDH}_p(0, g)$ to compute g^k, retrieve $\langle \mathsf{S}, \mathcal{SI}, p \rangle$ and record (p, g, g^k).
- In step 7, query $\mathsf{T\text{-}OMDH}_p(i, a)$ to compute k_i for honest server S_i. Set q_i++.
- In step 8, as soon as b_i for all $\mathsf{S} \in \mathcal{SE}$ defined in a record $\langle ssid, \mathsf{C}, \mathcal{SE}, g_d \rangle$ have been received from $\mathcal{A}$, compute $b := \prod_{\mathsf{S} \in \mathcal{SE}} b_i$, $\underline{\text{record } (g_d, b)}$ and find record (p^*, g_d^*, b^*) such that $\mathsf{DDH}(g_d, b, g_d^*, b^*)$. If there is no such record, choose a unique label p^* and record (p^*, g_d, b). Regardless, send $(\mathsf{RcvComplete}, sid, ssid, \mathsf{C}, p^*)$ to $\mathcal{F}$.
- In case (a) of step 9 (i.e., there is a record (x, g_d)), find record (p^*, g_d^*, b^*) such that $\mathsf{DDH}(g_d, u, g_d^*, b^*)$. If there is such a record, $\underline{\text{record } (g_d, u)}$. If there is no such record, choose a unique label p^*. After that, proceed as SIM.
  Finally, if $\mathsf{fail}(\tilde{\mathsf{S}})$ occurs, output the set of all underlined recorded pairs.

Figure 3.9: The reduction $\mathcal{R}$ to the Gap T-OMDH problem (for a single *sid*)

- Every time the $\mathsf{T\text{-}OMDH}_p(i,\cdot)$ oracle is queried (in step 7), there is a $(\mathsf{SvrComplete}, sid, ssid, \mathsf{S}_i)$ message, so $\mathsf{tickets}(p, \mathsf{S}_i)$ increments.
- Every time $\mathsf{tickets}(p, \mathsf{S}_i)$ decrements (in step 8 and step 9), a pair (g_d, g_d^k) is recorded (as underlined).

Therefore, let Q be the total number of $\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)$ queries made by $\mathcal{R}$, which corresponds to the number of $(\mathsf{SvrComplete}, sid, ssid, \mathsf{S}_i)$ messages, and Q' be the total number of triples recorded. Event fail suggests that Q' has exceeded $C_{t-t'+1}(\vec{q})$, violating the Gap T-OMDH assumption. □

3.4 UC Security Definition of PPSS

Password-Protected Secret Sharing (PPSS) was defined in [8] as (password-protected) secret-sharing of an *arbitrary message* and re-defined in [43] as protecting a *random key*. Both definitions are in the game-based setting. Here we define a UC notion of PPSS as a secure realization of an ideal PPSS functionality $\mathcal{F}_{\text{PPSS}}$ presented in Figure 3.10. The PPSS functionality we define is weaker than the UC PPSS functionality of [22] (called T-PASS therein), since it obviates the need for extracting malicious clients' inputs at the time the PPSS reconstruction protocol takes place. In order to avoid requiring such online input extraction we use a "ticketing mechanism" which is similar to the one we use in the UC definition of OPRF in Section 3.2. We believe that any protocol that realizes the T-PASS functionality of [22] should also realize our functionality $\mathcal{F}_{\text{PPSS}}$ of Figure 3.10; however, our functionality can also be implemented by a much more lightweight protocol that most likely do not realize the T-PASS functionality of [22]. Functionality $\mathcal{F}_{\text{PPSS}}$ we propose has three interfaces, initialization, reconstruction, and password test, on which we elaborate below.

The *initialization* command Init represents a client with a unique username, represented by

For every server S, initialize $\mathsf{tickets}(\mathsf{S})$ to 0; initialize $\mathsf{tested}(\mathsf{pw})$ to $\varnothing$.

Initialization

- On $(\mathsf{Init}, sid, \mathcal{SI}, \mathsf{pw})$ where $|\mathcal{SI}| = n$ from C, if this is the first Init message for sid, pick $K \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$, record $\langle \mathsf{Init}, \mathsf{C}, \mathcal{SI}, \mathsf{pw} \rangle$ and send $(\mathsf{Init}, sid, \mathsf{C}, \mathcal{SI})$ to $\mathcal{A}^*$. Also, if $|\mathcal{SI} \cap \mathsf{CorrSrv}| \geq t+1$, then send (K, pw) to $\mathcal{A}^*$.
- On $(\mathsf{SvrInit}, sid, \mathsf{S})$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{Init}, \mathsf{C}, \mathcal{SI}, \mathsf{pw} \rangle$ and $\mathsf{S} \in \mathcal{SI}$, then mark S as active and send $(\mathsf{SvrInit}, sid)$ to S.
- On $(\mathsf{CltInit}, sid)$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{Init}, \mathsf{C}, \mathcal{SI}, \mathsf{pw} \rangle$ and all servers in $\mathcal{SI}$ are marked active, then augment the record to $\langle \mathsf{Init}, \mathsf{C}, \mathcal{SI}, \mathsf{pw}, K \rangle$ and send $(\mathsf{CltInit}, sid, K)$ to C.

Reconstruction

- On $(\mathsf{Rec}, sid, ssid, \mathcal{SR}, \mathsf{pw}')$ where $|\mathcal{SR}| = t+1$ from C', if this is the first Rec command for $ssid$, retrieve $\langle \mathsf{Init}, \mathsf{C}, \mathcal{SI}, \mathsf{pw}, K \rangle$, record $\langle \mathsf{Rec}, ssid, \mathsf{C}', \mathcal{SI}, \mathcal{SR}, \mathsf{pw}, \mathsf{pw}' \rangle$ and send $(\mathsf{Rec}, \mathsf{C}', sid, ssid, \mathcal{SR})$ to $\mathcal{A}^*$.
- On $(\mathsf{SvrRec}, sid, ssid, \mathsf{S})$ from $\mathcal{A}^*$, if S is marked active, then set $\mathsf{tickets}(\mathsf{S})$++ and send $(\mathsf{SvrRec}, sid, ssid)$ to S.
- On $(\mathsf{CltRec}, sid, ssid, \mathcal{SC}, \mathsf{flag}, \mathsf{pw}^*, K^*)$ where $|\mathcal{SC}| = t+1$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{Rec}, ssid, \mathsf{C}', \mathcal{SI}, \mathcal{SR}, \mathsf{pw}, \mathsf{pw}', K \rangle$ such that $\mathcal{SR} \setminus \mathsf{CorrSrv} \subseteq \mathcal{SC}$ and $\mathsf{tickets}(\mathsf{S}) > 0$ for all $\mathsf{S} \in \mathcal{SC}$, then set $\mathsf{tickets}(\mathsf{S})$−− for all such S and do:
  - If $\mathsf{pw}' = \mathsf{pw} \wedge \mathcal{SC} \subseteq \mathcal{SI} \wedge (\mathsf{flag} = 1 \vee \mathcal{SR} \cap \mathsf{CorrSrv} = \varnothing)$, then set $\mathsf{Res} := K$.
  - If $\mathsf{pw}' = \mathsf{pw}^* \wedge \mathcal{SC} \subseteq \mathsf{CorrSrv} \wedge \mathsf{flag} = 2$, then set $\mathsf{Res} := K^*$.
  - Else set $\mathsf{Res} := \mathsf{fail}$.

  Finally, send $(\mathsf{CltRec}, sid, ssid, \mathsf{Res})$ to C'.

Password Test

- On $(\mathsf{TestPwd}, sid, \mathsf{S}_i, \mathsf{pw}^*)$ from $\mathcal{A}^*$, ignore this message if $\mathsf{tickets}(\mathsf{S}_i) = 0$. Else set $\mathsf{tested}(\mathsf{pw}^*) := \mathsf{tested}(\mathsf{pw}^*) \cup \{\mathsf{S}_i\}$ and $\mathsf{tickets}(\mathsf{S}_i)$−−, retrieve $\langle \mathsf{Init}, \mathsf{C}, \mathcal{SI}, \mathsf{pw}, K \rangle$, and if $|\mathcal{SI} \cap (\mathsf{tested}(\mathsf{pw}^*) \cup \mathsf{CorrSrv})| \geq t+1$, then return K to $\mathcal{A}^*$ if $\mathsf{pw}^* = \mathsf{pw}$ and fail otherwise.

Figure 3.10: Functionality $\mathcal{F}_{\mathrm{PPSS}}$ for parameters (t, n)

sid, at some network entity C initializing a PPSS instance with a set of n servers $\mathcal{SI} = \{\mathsf{S}_1, \ldots, \mathsf{S}_n\}$ on input a password pw. The servers in $\mathcal{SI}$ become activated for this instance, provided that the ideal-world adversary $\mathcal{A}^*$ agrees by sending command SvrInit, and if all servers in $\mathcal{SI}$ are activated the instance generates a random secret K output by C if $\mathcal{A}^*$ agrees by sending command CltInit. (In the real protocol this corresponds to adversary forwarding protocol messages correctly.) If set $\mathcal{SI}$ contains $t+1$ corrupted servers then $\mathcal{A}^*$ receives (pw, K), which corresponds to C creating a PPSS among n servers of whom more than t are corrupted, at which point a (t, n)-threshold secret-sharing no longer protects its secrets.

The *reconstruction* command Rec represents a client at a potentially different network entity C' attempting to recover the secret initialized above using password pw', which might or might not equal to pw above. The reconstruction operation is directed to some set of $t+1$ servers $\mathcal{SR}$, which might or might not overlap with set $\mathcal{SI}$ above. It is important to emphasize that the client maintains no state between the initialization and the reconstruction operations except for memorizing password pw (and username *sid*), although a failure to remember pw correctly is also allowed, and it is modeled by setting $\mathsf{pw}' \neq \mathsf{pw}$. In particular, the client might connect to a different set of servers in the initialization and in the reconstruction. Hence, for example, if a client falls prey to a phishing attack, she could execute the reconstruction protocol with servers $\mathcal{SR}$ such that $\mathcal{SR} \cap \mathcal{SI} = \varnothing$ and all servers in $\mathcal{SR}$ are corrupted. However, by the rules of the $\mathcal{F}_{\text{PPSS}}$ functionality which we explain below, the worst thing that can happen in this case is the inevitable online guessing attack: The adversary can execute the reconstruction protocol on behalf of the corrupt servers $\mathcal{SR}$ for some chosen password pw^* and secret key K^*, and in the case $\mathsf{pw}^* = \mathsf{pw}$ it would cause C to reconstruct K^* instead of K (or $\perp$).

SvrRec and CltRec commands control the view of the reconstruction protocol by the servers and the client in a similar way as in the Init above, but with some significant differences. First, C's sessions with any corrupt server in $\mathcal{SI}$ can be "routed" by $\mathcal{A}^*$ to any other server,

hence in CltRec command $\mathcal{A}^*$ specifies a set $\mathcal{SC}$ of servers who effectively participate in this reconstruction, with the only constrain that $\mathcal{SC}$ must contain all uncorrupted servers in $\mathcal{SI}$. Secondly, client's completion can result in two different outcomes (in addition to failure which $\mathcal{A}^*$ can except in the case when $\mathsf{pw}' = \mathsf{pw}$ and $\mathcal{SR}$ contains only uncorrupted servers in $\mathcal{SI}$): The default case is that the reconstruction works and C' outputs key K created in the initialization, which happens when $\mathsf{pw}' = \mathsf{pw}$, i.e., C' ran on the correct password, $\mathcal{SC} \subseteq \mathcal{SI}$, i.e., C' connected to servers participating in the initialization, and there were either no corrupt servers in the set $\mathcal{SR}$ with which C' attempted the reconstruction, or, if some of those servers are corrupted, $\mathcal{A}^*$ still allowed the protocol to succeed by setting the flag variable to 1. Another case is when C' connected only to corrupt servers (and $\mathcal{A}^*$ does not route these connections to uncorrupted servers, hence we require not only that $\mathcal{SR} \subseteq \mathsf{CorrSrv}$ but also that $\mathcal{SC} \subseteq \mathsf{CorrSrv}$, which is stronger because $\mathcal{SR} \setminus \mathsf{CorrSrv} \subseteq \mathcal{SC}$), because such reconstruction session offers $\mathcal{A}^*$ an ability to perform an *online guessing attack on the client*, i.e., $\mathcal{A}^*$ can specify its password guess pw^* and, in case $\mathsf{pw}^* = \mathsf{pw}$, cause C' to output any value K^* specified by $\mathcal{A}^*$. Indeed, a PPSS protocol which, like ours, does not assume any other source of client's security apart of the password, and in particular does *not* assume PKI for security, cannot offer stronger protection in the case the client executes the protocol with an adversary who guesses her password.

Finally, the *test password* command TestPwd lets the adversary $\mathcal{A}^*$ perform an online guessing attack on the servers, i.e., specify a password guess pw^* and a set S of at least $t+1$ servers in $\mathcal{SI}$, and learn key K if $\mathsf{pw}^* = \mathsf{pw}$. However, $\mathcal{F}_{\text{PPSS}}$ allows such guessing attack to proceed only if $\mathcal{A}^*$ engages the servers in S in reconstruction protocol instances for username *sid*, as represented by SvrRec commands. Every time such command is issued for some server S, functionality $\mathcal{F}_{\text{PPSS}}$ increments a *ticket counter* $\mathsf{tickets}(\mathsf{S})$, and the adversary can make a password guess only $\mathsf{tickets}(\mathsf{S}) > 0$ for all $\mathsf{S} \in S$, and the counters of serves in S are decremented as result of such password-testing attempt. Since a PPSS server cannot tell if a reconstruction protocol instance is originated by an honest client or by the adversary, any

such reconstruction session can be used *either* for completion of honest client reconstruction instances or for instances executed by the adversary. However, the ticket-counting mechanism of $\mathcal{F}_{\text{PPSS}}$ enforces that any PPSS instance completed by $t+1$ can be "used up" either for a single instance of the honest client reconstructing its secret or for a single instance of an adversary who attempts the reconstruction on a guessed password pw^*.

3.5 TOPPSS: A PPSS Protocol Based on T-OPRF

In Figure 3.11 we show a compiler which converts a T-OPRF protocol which realizes the UC T-OPRF notion in Section 3.3 into a PPSS protocol, called TOPPSS, which realizes the UC PPSS functionality $\mathcal{F}_{\text{PPSS}}$ in Section 3.4.

Overview. To explain the mechanics of TOPPSS based on the T-OPRF functionality, it is instructive to compare it to the OPRF-based PPSS protocol of [43]. In that protocol each server holds its own *independently random* key k_i for an OPRF F. At initialization, the secret to be protected is processed with a (t, n) secret sharing scheme and each share is stored at one of n servers, where server S_i stores the i-th share encrypted under $F_{k_i}(\mathsf{pw})$. At reconstruction, the client receives the encrypted shares from $t+1$ servers which it decrypts using the values $F_{k_i}(\mathsf{pw})$ that it learns by running the OPRF on pw with each of these servers. By contrast, in our TOPPSS protocol, which is T-OPRF-based, the (random) secret to be protected is defined as a single PRF value $v = F_k(\mathsf{pw})$ where k is a key secret-shared as part of a T-OPRF protocol. This provides a significant performance gain by reducing the number of exponentiations performed by the client from $t+2$ to just 2. In the protocol of [43] implemented with 2HashDH, the client computes the OPRF sub-protocol with each server independently, which involves one blinding operation re-used across all servers, but requires one de-blinding operation *per server* for a total of $t+2$ exponentiations. By contrast, in the

Let $H(\cdot)$ be a hash function with range $\{0,1\}^{2\kappa}$ (modeled as a random oracle).
Initialization (assume an authenticated channel)

1. On input $(\mathsf{Init}, sid, \mathcal{SI}, \mathsf{pw})$, C sends $\mathcal{SI}$ to all $\mathsf{S} \in \mathcal{SI}$.
2. On $\mathcal{SI}$ from C, S sends $(\mathsf{Init}, sid, \mathcal{SI})$ to $\mathcal{F}_{\text{TOPRF}}$.
   On $\mathcal{F}_{\text{TOPRF}}$'s response $(\mathsf{InitComplete}, sid)$, S sends done to C.
3. On done from all $\mathsf{S} \in \mathcal{SI}$, C sends $(\mathsf{Eval}, sid, 0, \mathcal{SE}, \mathsf{pw})$ to $\mathcal{F}_{\text{TOPRF}}$ for any $\mathcal{SE} \subseteq \mathcal{SI}$ where $|\mathcal{SE}| = t + 1$.
   On $\mathcal{F}_{\text{TOPRF}}$'s response $(\mathsf{Eval}, sid, 0, v)$, C parses $H(v)$ as $[C||K]$ and sends C to all $\mathsf{S} \in \mathcal{SI}$.
4. On C from C, S records C, sends ack to C and outputs $(\mathsf{SvrInit}, sid)$.
5. On ack from all $\mathsf{S} \in \mathcal{SI}$, C outputs $(\mathsf{CltInit}, sid, K)$.

Reconstruction

1. On input $(\mathsf{Rec}, sid, ssid, \mathcal{SR}, \mathsf{pw}')$, C sends $(\mathsf{Eval}, sid, [1||ssid], \mathcal{SR}, \mathsf{pw}')$ to $\mathcal{F}_{\text{TOPRF}}$.
2. On $(\mathsf{SvrComplete}, sid, [1||ssid])$ from $\mathcal{F}_{\text{TOPRF}}$, if S holds record C, then it sends C to C and outputs $(\mathsf{SvrRec}, sid, ssid)$.
3. On $\mathcal{F}_{\text{TOPRF}}$'s response $(\mathsf{Eval}, sid, [1||ssid], v')$ and C' from all $\mathsf{S} \in \mathcal{SR}$, C outputs $(\mathsf{CltRec}, sid, ssid, \mathsf{Res})$ where Res is defined as follows:
   - If each message contains C' such that $[C'||K'] = H(v')$, then C sets $\mathsf{Res} := K'$.
   - Else C sets $\mathsf{Res} := \mathsf{fail}$.

Figure 3.11: The TOPPSS protocol in the $\mathcal{F}_{\text{TOPRF}}$-hybrid world

T-OPRF protocol 2HashTDH the client performs a single blinding and de-blinding, hence just 2 exponentiations, regardless of the number of servers and threshold t.

Note that the T-OPRF functionality allows the client to evaluate function $F_k(\cdot)$ on the client's password pw, without leaking any information about pw, but it does not let the client verify whether the function is computed correctly. Indeed, following the rules of functionality $\mathcal{F}_{\text{TOPRF}}$, either corrupt servers or a man-in-the-middle adversary could make the client compute $F_k(\mathsf{pw})$ on key k of its choice. If the dictionary D from which the client draws her password is small, the adversary can potentially pick k such that function $F_k(\cdot)$ behaves on domain D in some ways the adversary can exploit (e.g., reducing the number of possible outputs). However, since $\mathcal{F}_{\text{TOPRF}}$ assures that $F_k(\cdot)$ behaves like a random function for all k's, even for k's chosen by the adversary, it suffices to include a commitment to the master secret $v = F_k(\mathsf{pw})$ in the information that the servers send to the client, so that the client can verify its correctness. The adversary can still pick k but if $F_k(\cdot)$ is pseudorandom for all k then the adversary cannot change either k or v without guessing pw. Note that the randomness for verifying this commitment must be derived from the committed plaintext $F_k(\mathsf{pw})$ itself as this is the only value the client can retrieve using its only input pw. Although this mechanism requires the commitment scheme to be deterministic, the hiding property of the commitment is still satisfied thanks to the pseudorandomness of the committed plaintext $v = F_k(\mathsf{pw})$ (and assuming no more than t corruptions).

Since our realizations of $\mathcal{F}_{\text{TOPRF}}$, protocol 2HashTDH, requires the ROM for hash functions in the security analysis, we implement this commitment simply with another hash function modeled as a random oracle. Finally, since the client needs to verify the master-secret v as well as to derive a key K from it, we implement both operation using a single hash function call, i.e., we set $[C||K]$ to $H(v)$ where $H(\cdot)$ hashes onto strings of length 2κ.

Theorem 5. *Protocol TOPPSS in Figure 3.11 realizes functionality $\mathcal{F}_{\text{PPSS}}$ in the $\mathcal{F}_{\text{TOPRF}}$-hybrid world in ROM.*

For every index p and every server S, initialize $\mathsf{tickets}(p, \mathsf{S})$ to 0; initialize $\mathsf{counter}$ to 0.
Initialization

1. On $(\mathsf{Init}, sid, \mathsf{C}, \mathcal{SI})$ from $\mathcal{F}$ where $|\mathcal{SI}| = n$, if this is the first Init message for sid, set $\mathsf{counter}{+}{+}$, record $\langle \mathsf{C}, \mathcal{SI}, \mathsf{counter} \rangle$ and send $\mathcal{SI}$ to $\mathcal{A}$ as a message from all $\mathsf{S} \in \mathcal{SI}$. Also, if $\mathcal{F}_{\text{PPSS}}$ sends (pw, K), record it.

2. On $\mathcal{SI}$ from $\mathcal{A}$ as a message from $\mathsf{S} \in \mathcal{SI}$, mark S active and send $(\mathsf{Init}, sid, \mathsf{S}, \mathcal{SI}, \mathsf{counter})$ to $\mathcal{A}$.

3. On $(\mathsf{InitComplete}, sid, \mathsf{S})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{TOPRF}}$ for some $\mathsf{S} \in \mathcal{SI}$, if there is a record $\langle \mathsf{C}, \mathcal{SI}, \mathsf{counter} \rangle$, and all servers in $\mathcal{SI}$ are marked active, send done to $\mathcal{A}$ as a message from S.

4. On done $\mathcal{A}$ as a message from all $\mathsf{S} \in \mathcal{SI}$, send $(\mathsf{Eval}, sid, 0, \mathsf{C}, \mathcal{SE})$ to $\mathcal{A}$ for any $\mathcal{SE} \subseteq \mathcal{SI}$ where $|\mathcal{SE}| = t + 1$.

5. On $(\mathsf{RcvComplete}, sid, ssid, \mathsf{P}, p^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{TOPRF}}$, retrieve $\langle \mathsf{C}, \mathcal{SI}, p \rangle$; ignore this message if there is no such record, or $p^* = p$ but $|\{\mathsf{S} \in \mathcal{SI} \mid \mathsf{tickets}(p, \mathsf{S}) > 0\}| \leq t$. Else pick $C \leftarrow_{\text{R}} \{0,1\}^{\kappa}$, augment $\langle \mathsf{C}, \mathcal{SI}, p \rangle$ to $\langle \mathsf{C}, \mathcal{SI}, p, C \rangle$ and send C to $\mathcal{A}$ as a message from C to all $\mathsf{S} \in \mathcal{SI}$.
   Also, if there is a record (pw, K) and $F_p(\mathsf{pw})$ is defined, then set $H(F_p(\mathsf{pw})) := [C||K]$; else pick $K \leftarrow_{\text{R}} \{0,1\}^{\kappa}$.
   Furthermore, if $p^* = p$, then set $\mathsf{tickets}(p^*, \mathsf{S})--$ for any $t + 1$ distinct S such that $\mathsf{tickets}(p, \mathsf{S}) > 0$.

6. On C from $\mathcal{A}$ for some $\mathsf{S} \in \mathcal{SI}$, send ack to $\mathcal{A}$ and $(\mathsf{SvrInit}, sid, \mathsf{S})$ to $\mathcal{F}$.

7. On ack from $\mathcal{A}$ as a message from $\mathsf{S} \in \mathcal{SI}$, mark S $\mathsf{completed}$. As soon as all servers in $\mathcal{SI}$ are marked as $\mathsf{completed}$, send $(\mathsf{CltInit}, sid, K)$ to $\mathcal{F}$.

Figure 3.12: The simulator SIM for TOPPSS in the initialization phase

Proof. For any efficient adversary $\mathcal{A}$ against the protocol, we construct a simulator SIM as in Figure 3.3. To keep notation brief we denote functionality $\mathcal{F}_{\text{PPSS}}$ as $\mathcal{F}$.

First, note that SIM assigns an $H(\cdot)$ value to a certain string in step 5 in initialization, and steps 3, 4 and 5 in reconstruction; if there is a conflict in such assignments, that is, when SIM assigns a value $H(\cdot)$ to a certain string in one of the above four steps, but it was assigned previously, SIM outputs halt. We show that $\Pr[\mathsf{halt}]$ is negligible:

- Step 5 in initialization: Here $H(F_p(\mathsf{pw}))$ is set to $[C||K]$ if at least $t + 1$ servers in $\mathcal{SI}$

Reconstruction

1. On $(\mathsf{Rec}, \mathsf{C}, sid, ssid, \mathcal{SR})$ from $\mathcal{F}$, record $\langle ssid, \mathsf{C}, \mathcal{SR}\rangle$ and send $(\mathsf{Eval}, sid, ssid, \mathsf{C}, \mathcal{SR})$ to $\mathcal{A}$.

2. On $(\mathsf{SvrComplete}, sid, ssid, \mathsf{S})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{TOPRF}}$, retrieve $\langle \mathsf{C}, \mathcal{SI}, p\rangle$, set $\mathsf{tickets}(p, \mathsf{S})$++ and send $(\mathsf{SvrRec}, sid, ssid, \mathsf{S})$ to $\mathcal{F}$.

3. On $(\mathsf{RcvComplete}, sid, ssid, \mathsf{C}, p^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{TOPRF}}$ and C' from $\mathcal{A}$ as a message to C from all $\mathsf{S} \in \mathcal{SR}$, retrieve $\langle ssid, \mathsf{C}, \mathcal{SR}\rangle$ and $\langle \mathsf{C}, \mathcal{SI}, p\rangle$; ignore this message if (i) there is no such record, or (ii) all servers in $\mathcal{SE}$ are honest but $p^* \neq p$, or (iii) $p^* = p$ but $|\{\mathsf{S} \in \mathcal{SI} \mid \mathsf{tickets}(p, \mathsf{S}) > 0\}| \leq t$. Else if $p^* = p$, then set $\mathsf{tickets}(p^*, \mathsf{S})$−− for any $t+1$ distinct S such that $\mathsf{tickets}(p, \mathsf{S}) > 0$, and send $(\mathsf{CltRec}, sid, \mathcal{SR}, \mathsf{flag}, \mathsf{pw}^*, K^*)$ to $\mathcal{F}$ where $(\mathsf{flag}, \mathsf{pw}^*, K^*)$ is defined as follows:

   (a) If not all C''s are the same, then set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (0, \perp, \perp)$.

   (b) Else retrieve $\langle \mathsf{C}, \mathcal{SI}, \mathsf{counter}\rangle$. If $p^* = p$ and $C' = C$, then set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (1, \perp, \perp)$.

   (c) Else for every x such that $F_{p^*}(x)$ is defined, set $v' := F_{p^*}(x)$ and check if $C' = H_L(v')$. If so, then set $K' := H_R(v')$ and $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (2, x, K')$ and break the loop. If there is no such x, then set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (0, \perp, \perp)$.

4. On $(\mathsf{Eval}, sid, ssid, \mathcal{SE}, x)$ from $\mathsf{P} \in \{\mathsf{C}, \mathcal{A}\}$ and $(\mathsf{RcvComplete}, sid, ssid, \mathsf{P}, p^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{TOPRF}}$, retrieve $\langle ssid, \mathsf{C}, \mathcal{SR}\rangle$ and $\langle \mathsf{C}, \mathcal{SI}, p\rangle$; ignore this message if there is no such record, or $p^* = p$ but $|\{\mathsf{S} \in \mathcal{SI} \mid \mathsf{tickets}(p, \mathsf{S}) > 0\}| \leq t$. Else send $(\mathsf{Eval}, sid, ssid, F_{p^*}(x))$ to $\mathcal{A}$ (pick $F_{p^*}(x) \leftarrow_{\text{R}} \{0,1\}^\kappa$ if undefined). If $p^* = p$, then also set $\mathsf{tickets}(p^*, \mathsf{S})$−− for any $t+1$ distinct S such that $\mathsf{tickets}(p, \mathsf{S}) > 0$, add every $\mathsf{S} \in \mathcal{SE}$ to $\mathsf{tested}(x)$ and send $(\mathsf{TestPwd}, sid, \mathsf{S}, x)$ to $\mathcal{F}$. If $\mathcal{F}$ returns K, then also set $H(F_{p^*}(x)) := [C||K]$.

5. Answer $\mathcal{A}$'s $H(\cdot)$ queries via lazy sampling unless described in previous steps. If there is a conflict in the assignment of $H(\cdot)$ values, output halt and abort.

Figure 3.13: The simulator SIM for TOPPSS in the reconstruction phase

are corrupted. Suppose in this step $H(F_p(\mathsf{pw}))$ is already assigned to another value (i) in step 3 in reconstruction: according to the syntax of $\mathcal{F}_{\mathrm{PPSS}}$, reconstruction cannot be proceeded before initialization, so this is impossible. (ii) in step 4 in reconstruction: if step 4 in reconstruction is proceeded before step 5 in initialization, there is no C found, and SIM will ignore C^* and $\mathcal{A}$'s message. So there will be no assignment. (iii) in step 5 in reconstruction: unless and until $\mathcal{A}$ queries $F_p(\mathsf{pw})$, $F_p(\mathsf{pw})$ is a random string in $\{0,1\}^\kappa$ in $\mathcal{Z}$'s view, so the probability that $\mathcal{Z}$ queries $H(F_p(\mathsf{pw}))$ is at most $1/2^\kappa$. Once $\mathcal{Z}$ queries $F_p(\mathsf{pw})$ (note that this query can be done only in step 4 in reconstruction), this case transfers to case (ii).

- Step 4 in reconstruction: Here $H(F_{p^*}(x))$ is set to $[C||K]$ if there are at least $t+1$ servers in either $\mathsf{tested}(x)$ or the corrupted server subset of $\mathcal{SI}$, and $x = \mathsf{pw}$. Suppose in this step $H(F_{p^*}(\mathsf{pw}))$ is already assigned to another value (i) in step 5 in initialization: note that in step 4 in reconstruction, C and K are exactly the same with those in step 5 in initialization, so there is no possibility of conflict. (ii) in step 3 in reconstruction: in this step, the computation of $F_{p^*}(\mathsf{pw})$ may occur in case (c), where $F_{p^*}(\mathsf{pw})$ is already defined. However, $F_{p^*}(\mathsf{pw})$ can be defined only through querying it in step 4 in reconstruction, and once it is queried, $F_{p^*}(\mathsf{pw})$ will be assigned to $[C||K]$ immediately. Therefore, it is impossible that SIM needs to set $F_{p^*}(\mathsf{pw})$ to some value after it has already been assigned in step 3 in reconstruction. (iii) in step 5 in reconstruction: similar to case (iii) in the bullet above, in this case fail occurs with probability at most $1/2^\kappa$.

- Steps 3 and 5 in reconstruction: These two cases are trivial, since here $H(\cdot)$ is assigned to a certain value only if it has not been set previously; that is, there is no possibility that $H(\cdot)$ is assigned again after it has been assigned to another value.

Below we assume that halt does not occur.

We now show that the distinguishing advantage of $\mathcal{Z}$ between the real world and the simulated world is negligible. As before, the argument uses a sequence of games, starting from the real world and ending at the simulated world.

$\mathbf{G}_0$ is the real world and shown in Figure 3.14, where we make a number of simplifications explained below:

- For all messages input from and output to $\mathcal{Z}$, the names of them (e.g., Init and Rec) and session IDs (i.e., *sid* and *ssid*) are omitted.
- There is no difference between the real world and the simulated world regarding done, ack (sent from S to C in initialization), and (SvrRec, *sid*, *ssid*) (output by S in reconstruction), so we omit these messages below.
- $\mathcal{Z}$'s input in $\mathcal{F}_{\text{TOPRF}}$ queries includes (Eval, *sid*, *ssid*, $\mathcal{SE}$, x) (sent to SIM via $\mathcal{A}$ or a corrupt client C^*) and (RcvComplete, *sid*, *ssid*, P, p^*) (sent to SIM via $\mathcal{A}$). The messages are ignored if one of the conditions listed as (i), (ii) and (iii) in Figure 3.13 holds; therefore, if one of those conditions holds, the case is trivial and we do not consider such cases below. Otherwise the output is computed as $F_{p^*}(\mathsf{pw}^*)$. Therefore, step 8 of SIM is essentially querying the F functions maintained by $\mathcal{F}_{\text{TOPRF}}$. Thus, we simplify the input to the function pointer p and the variable x, and the output to the function value $v = T(p, x)$, and omit all other messages and entries exchanged among the participating parties.
- $\mathcal{Z}$'s view in reconstruction includes messages (Rec, [...], pw′) output by C, (SvrComplete, [...]) output by S, and (RcvComplete, [...], p^*) output by C. As in the bullet above, if the whole process ends with C outputting either a string $K' \in \{0,1\}^\kappa$ or fail, then $\mathcal{SR}$ is not related to the final result. Therefore, we do not show them below, and simplify $\mathcal{Z}$'s input to pw′ and C'. Furthermore, C outputs fail immediately if it receives two different C''s, so this case is trivial. We only consider the other case,

i.e., all C''s are the same, in the games.

Set $F_p(\cdot)$ to a random function with range $\{0,1\}^\kappa$.
Initialization
On inputs $\mathcal{SI}$ and pw:

1. Set p to a fresh non-negative integer and compute $v := F_p(\mathsf{pw})$.
2. Output $K := H_L(v)$ and $C := H_R(v)$.

Reconstruction
On inputs p^*, pw' and C':

1. Set $v' := F_{p^*}(\mathsf{pw}')$.
2. Check if $C' = H_L(v')$. If so, output $K' := H_R(v')$; else output fail.

$\mathcal{F}_{\text{TOPRF}}$ Queries
On inputs p^* and x, output $F_{p^*}(x)$.
Hash Function Queries
On input x, output $H(x)$.

Figure 3.14: Game $\mathbf{G}_0$ in the security proof for TOPPSS (simplified)

In $\mathbf{G}_1$, reconstruction is proceeded as follows:

- If $p^* = p$ and $C' = C$ (we denote such event as E_C below), then output K to $\mathcal{Z}$ if $\mathsf{pw}' = \mathsf{pw}$ and fail otherwise.
- Else let X be the set of all x in the dictionary such that $F_{p^*}(x)$ is queried. Iterate through all $x \in X$ in lexicographic order and perform reconstruction as in $\mathbf{G}_0$, except that pw' is replaced with x; that is, compute $v' := F_{p^*}(x)$, check if $C' = H_L(v')$, and if so,
  - If $x = \mathsf{pw}'$, then output $K' := H_R(v')$.
  - Else output fail.

  In either case, break the loop. If the loop ends without a break (i.e., the check does not pass for every $x \in X$), output fail.

We compare $\mathcal{Z}$'s views in $\mathbf{G}_1$ and $\mathbf{G}_0$. Let K'_1 and K'_0 be the output at the end of reconstruction in $\mathbf{G}_1$ and $\mathbf{G}_0$, respectively. Let event E be $K'_1 \neq K'_0$.

First, note that if $K'_0 = \mathsf{fail}$, then $K'_1 = \mathsf{fail}$. This is equivalent to if $K'_1 \neq \mathsf{fail}$, then $K'_0 \neq \mathsf{fail}$. This is because: (1) If E_C occurs and $K'_1 \neq \mathsf{fail}$, this means that $\mathsf{pw}' = \mathsf{pw}$, in which case $C = H_L(F_p(\mathsf{pw}))$, so the check in $\mathbf{G}_0$ passes; that is, $K'_0 \neq \mathsf{fail}$. (2) If E_C does not occur and $K'_1 \neq \mathsf{fail}$, then $C' = H_L(T(p^*, \mathsf{pw}'))$, so the check in $\mathbf{G}_0$ passes; that is, $K'_0 \neq \mathsf{fail}$. Therefore, E can only occur when $K'_0 \neq \mathsf{fail}$.

Next we break E into several sub-events:

- E_1: $E_C \wedge K'_1 \neq K'_0$.

  In this case, $\mathsf{pw}' \neq \mathsf{pw}$ must hold (otherwise $K'_1 = K'_0 = K$ where K is the output in initialization of $\mathbf{G}_1$ and $\mathbf{G}_0$), and since $K'_0 \neq \mathsf{fail}$, the check in $\mathbf{G}_0$ passes; that is, $C' = C = H_L(v')$. On the other hand, we know from initialization that $C = H_L(v)$. Note that $v' = F_p(\mathsf{pw}')$, $v = F_p(\mathsf{pw})$, and $\mathsf{pw}' \neq \mathsf{pw}$; therefore, if E_1 occurs, $(\mathsf{pw}, \mathsf{pw}')$ forms a pair of collision of $H_L(F_p(\cdot))$, which is a random function with range $\{0,1\}^\kappa$. Assuming that there are q_C C sub-sessions, there is one pw and q_C pw''s, so we have that $\Pr[E_1] \leq q_\mathrm{C}/2^\kappa$.

- E_2: $\neg E_C \wedge \mathsf{pw}' \notin X \wedge K'_1 \neq K'_0$.

  First consider the case that $p^* = p$ and $\mathsf{pw}' = \mathsf{pw}$. If so, since $K'_0 \neq \mathsf{fail}$, we have that $C' = H_L(F_{p^*}(\mathsf{pw}')) = H_L(F_p(\mathsf{pw})) = C$. However, since E_C does not occur and $p^* = p$, $C' \neq C$ must hold, which contradicts the former. Therefore, if E_2 occurs, either $p^* \neq p$ or $\mathsf{pw}' \neq \mathsf{pw}$. In either case, since $\mathsf{pw}' \notin X$, that is, $F_{p^*}(\mathsf{pw})$ is not queried, $F_{p^*}(\mathsf{pw})$ is a random string in $\{0,1\}^\kappa$ in $\mathcal{Z}$'s view. Thus, for a single C', $\Pr[C' = H_L(F_{p^*}(\mathsf{pw}))] \leq 1/2^\kappa$. Assuming that there are q_S S sub-sessions, we have that $\Pr[E_2] \leq q_\mathrm{S}/2^\kappa$.

- E_3: $\neg E_C \wedge \mathsf{pw}' \in X \wedge K'_1 \neq K'_0$.

In this case, $\mathbf{G}_1$ will search X in lexicographic order, and once it comes to pw', it will find out that $C' = H_L(F_{p^*}(\mathsf{pw}'))$ (this is implied by $K'_0 \neq \mathsf{fail}$) and output $K'_1 := H_R(F_{p^*}(\mathsf{pw}')) = K'_0$. Therefore, if E_3 occurs, there exists $x < \mathsf{pw}'$ such that $C' = H_L(F_{p^*}(x))$. But $C' = H_L(F_{p^*}(\mathsf{pw}'))$, so (x, pw') forms a pair of collision of $H_L(F_{p^*}(\cdot))$. Assuming that F is queried q_{F} times, we have that $\Pr[E_3] \leq q_{\mathrm{C}} q_{\mathrm{F}} / 2^{\kappa}$.

We conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_0,\mathbf{G}_1} \leq \Pr[E] \leq \Pr[E_1] + \Pr[E_2] + \Pr[E_3] \leq \frac{q_{\mathrm{F}}(q_{\mathrm{C}} + 1) + q_{\mathrm{S}}}{2^{\kappa}},$$

which is a negligible function of κ. Let $\mathbf{G}_2$ be a modification of $\mathbf{G}_1$, where in initialization, pick $[C||K] \leftarrow_{\mathrm{R}} \{0,1\}^{2\kappa}$, and once $F_p(\mathsf{pw})$ is queried, set $H(F_p(\mathsf{pw})) := [C||K]$. We can see that $\mathbf{G}_2$ is essentially the same with the security game in the simulated world.

In $\mathbf{G}_1$, before $F_p(\mathsf{pw})$ is queried, it is random in $\mathcal{Z}$'s view, so the probability that $\mathcal{Z}$ queries $H(F_p(\mathsf{pw}))$ is negligible. If $\mathcal{Z}$ does not query $H(F_p(\mathsf{pw}))$, C and K are random strings in $\mathcal{Z}$'s view; that is how they are generated in $\mathbf{G}_2$. After $F_p(\mathsf{pw})$ is queried, $\mathbf{G}_2$ are $\mathbf{G}_1$ are identical. Therefore, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_1,\mathbf{G}_2} = 0.$$

Summing up all results above, we conclude that $\mathcal{Z}$'s distinguishing advantage between the real world and the simulated world is a negligible function of κ. This completes the proof. $\square$

Chapter 4

A Round-Reduced Modular Construction of Asymmetric Password-Authenticated Key Exchange

In this chapter we show a new compiler which converts any UC secure *symmetric* PAKE protocol into a UC secure *asymmetric* PAKE. Our construction relies on ROM, as do all UC asymmetric PAKE protocols proposed so far [40, 49], and the Computational Diffie-Hellman (CDH) assumption. Our starting point is the Ω-method due to Gentry *et al.* [40], which transforms any UC PAKE protocol into a UC aPAKE secure in ROM. The main point of our compiler is that they add only a *single additional message* to the underlying PAKE, in contrast to the Ω-method which adds two messages.

The results presented in this chapter are based on the work published in [42], with the security proof added. [42] includes two general PAKE-to-aPAKE compilers, and we only

include one of them here.

4.1 Overview

Round complexity of our compiler v. the Ω-method compiler. The Ω-method compiler adds (up to) two communication rounds to the underlying PAKE. In constract, our PAKE-to-aPAKE compiler adds only a *single additional message*. Moreover, the single extra message in our PAKE-to-aPAKE compiler is sent from client to server, and therefore in an application where the aPAKE instance, which establishes a secure session key for both parties, is followed by an explicit client-to-server entity authentication, e.g., the client uses the session key output by PAKE to send a MAC on the aPAKE transcript to the server, this additional message can be piggybacked with the client's explicit entity authentication flow. Likewise, if the last message of the symmetric UC PAKE is client-to-server, our compilers also add no additional communication flow to the protocol. By contrast, the Ω-method would add 2 message flows in the latter case.

We note that if the last round in the symmetric UC PAKE was server-to-client, then our compilers would offer no advantage over the Ω-method: Our compilers would add one client-to-server round, and so would the Ω-method because its first message c' (see Figure 1 in [40]) would be piggybacked on the last server-to-client flow of the underlying UC PAKE. Moreover, note that the symmetric UC PAKE, by its very nature, has no fixed roles, therefore every UC PAKE protocol can be executed so that the last message flow is server-to-client. Indeed, if the underlying n-round UC PAKE is executed in this way then the UC aPAKE resulting from both our compilers and the Ω-method would have $n+1$ rounds, with the last flow being client-to-server. However, the optimal way to arrange the n-round UC PAKE for the purpose of our compiler is so that its last flow is client-to-server, in which case our compilers output n-round UC aPAKE, while the Ω-method outputs an $(n+2)$-round UC aPAKE. Finally,

note that sometimes one will not have the flexibility of arranging the underlying PAKE in a way that optimizes the resulting aPAKE, because sometimes the choice of party who starts the interaction, i.e., whether it is the client or the server, will be fixed by an application.

Computational cost of our compiler vs. the Ω-method compiler. The computation overhead of the Ω-method compiler is dominated by a signature generation for the client and signature verification for the server. Instantiated with ECDSA signatures, both of these costs are only 1 (multi-)exponentiation per party.

However, since the Ω-method is a compiler, the exact costs of UC aPAKE it produces depend on the costs of the UC PAKE with which it is instantiated. While there is very active research on standard-model UC PAKEs, including round-minimal PAKEs [52, 39, 53, 48], these constructions are typically more expensive and require stronger assumptions than protocols satisfying game-based PAKE notions [10, 19] in ROM. Since any UC aPAKE construction requires non-black-box assumptions (see below), it makes sense to instantiate the Ω-method with a UC PAKE secure in ROM. However, while there are many 2-round game-based PAKEs whose cost is close to (intuitively minimal) 2 exponentiations per party of Diffie-Hellman Key Exchange (see e.g., [6] and references therein), we know of only one UC PAKE with comparable efficiency, by Abdalla *et al.* [2], which relies on the DDH assumption in ROM and the Ideal Cipher (IC) model, and uses 2 exponentiations per party. Combined with the Ω-method of Gentry *et al.*, the UC *symmetric* PAKE of [2] implies a UC *asymmetric* PAKE with 3 exponentiations per party, secure under the DDH assumption in ROM+IC model.

In contrast, the computational costs of our compiler is 1 exponentiation per client and 2 per server, which is slightly higher than that of the Ω-method complier. Looking a little closer, the costs of each option can be affected by the fact that in our compiler, the client's exponentiation is variable-base and the two server's exponentiations are fixed-base, with one

base fixed globally and the second base fixed per each user account.

Related work. We know of only one further UC aPAKE constructions in addition to the Ω-method of [40]. Jutla and Roy [49] proposed a round-minimal UC aPAKE in ROM, i.e., client and server send a single message and they can do so simultaneously, but their protocol requires groups with bilinear maps, uses significantly more exponentiations (and bilinear maps) per party.

We note that Benhamouda and Pointcheval [14] upgraded the game-based definition of aPAKE, called *verifier-based PAKE* therein, by strengthening the game-based aPAKE model of [19] to arbitrary password distributions and related passwords. One point of strengthening game-based aPAKE notion given that a UC aPAKE notion exists is a potential for better efficiency, but the other is that the UC aPAKE model of [40] seems not to be realizable without some non-black-box assumption on the adversary's local computation, like ROM, IC, or a generic group model. Indeed, the UC aPAKE model requires the simulator to extract offline password tests from the adversary's local computation of the hash function applied to password guesses. However, [14] relies on the *tight one-wayness* requirement on the hash function applied to passwords when creating the hashed password on the server, namely that given hash of a password chosen with δ min-entropy, the adversary has to compute 2^δ hash function instances to find it. Unfortunately, this notion also seems impossible to realize without similar non-black-box assumptions on the adversary, and [14] also relies on ROM to argue that this property is satisfied. Regarding computational costs, by avoiding random oracles on the protocol level (but not on the level of the underlying hash function), the aPAKE's of [19] are significantly more expensive than the UC aPAKE resulting from [40, 2]. Their 2-round protocol uses significantly more exponentiations per party, and the 1-round protocol requires groups with bilinear maps and has a still higher local computation cost.

4.2 Security Model

Our protocols convert any UC secure symmetric PAKE into a UC secure *asymmetric* PAKE, exactly like the protocol of [40], and we assume the same models of UC (revised) symmetric PAKE and asymmetric PAKE as in [40], denoted $\mathcal{F}_{\text{rPAKE}}$ and $\mathcal{F}_{\text{aPAKE}}$, respectively. (In [40] they were denoted $\mathcal{F}_{\text{rpwKE}}$ and $\mathcal{F}_{\text{apwKE}}$, respectively.) For completeness we include the full description of both functionalities in Figure 4.1, Figure 4.2 and Figure 4.3. Below we sketch the most important points in which these functionalities differ from the standard UC PAKE functionality of [24], and we refer to [40] for their full exposition.

The revised PAKE functionality $\mathcal{F}_{\text{rPAKE}}$. The symmetric PAKE functionality $\mathcal{F}_{\text{rPAKE}}$ defined by [40] is a revision of the original PAKE functionality $\mathcal{F}_{\text{PAKE}}$ defined by Canetti *et al.* [26]. Namely, it allows the functionality to produce a bitstring representing a *transcript* of the real-world execution of the PAKE protocol. Clearly, every real-world protocol has a transcript, but a typical UC functionality is concerned only with its "functional" input/output behavior and often omits the fact that various "objects" involved in protocol operation, e.g., private keys, public keys, transcripts, have physical encodings as bitstrings. This is unfortunate (and it is often not easy to do) because in protocol composition it can be very useful to process such objects through other cryptographic mechanisms, e.g., to sign them, encrypt them, secret-share them, etc. The idea of the PAKE-to-aPAKE compiler of Gentry *et al.* [40] was for the client to sign the PAKE transcript using a key encrypted by the server using the session key output by the symmetric PAKE. This signature acts in the Gentry *et al.* construction as a proof of possession of the password. However, for this modular construction to work, the UC symmetric PAKE functionality must expose some bitstring as the transcript to the environment. This is the sole point of the revised UC PAKE functionality $\mathcal{F}_{\text{rPAKE}}$ compared to the one defined in [26], and we adopt this revision because our compilers will likewise use the transcript of the symmetric

PAKE to bind the proof-of-password-possession to the underlying symmetric PAKE instance, although we will implement this proof-of-password-possession using different cryptographic mechanisms than the encrypted-key/signature-on-transcript protocol of Gentry *et al.*

The Asymmetric PAKE Functionality $\mathcal{F}_{\text{aPAKE}}$. The asymmetric functionality $\mathcal{F}_{\text{aPAKE}}$ is a more fundamental modification of the symmetric PAKE functionality $\mathcal{F}_{\text{PAKE}}$ [26], which models password authentication in the setting where only one party, the client, authenticates using a password, while the other, the server, uses a *password file*, which without loss of generality is an output of some (randomized) one-way function applied to the password during the initialization procedure. For example, in the standard password-over-TLS implementation the password file is a pair consisting of a random nonce known as *salt* and a hash of the password concatenated with this salt value. In the $\mathcal{F}_{\text{aPAKE}}$ functionality, creation of the password file on the server is modeled by command StorePwdFile, and note that the server-side invocation of the authentication protocol instance, via command SvrSession, does not take the password as an input, because its implicit input is the stored password file corresponding to session ID *sid* of this aPAKE instance. (It is assumed that a unique *sid* would be assigned to each user account held by a given server.)

The other fundamental difference between the asymmetric PAKE functionality $\mathcal{F}_{\text{aPAKE}}$ and the symmetric PAKE functionality $\mathcal{F}_{\text{PAKE}}$ is that an adversary may adaptively compromise the server and learn the stored password file, which is modeled by query StealPwdFile. Such adaptive server compromise allows the adversary to then impersonate the server to the client, modeled via the Impersonate command, because a real-world adversary could use the stolen password file to emulate the server in the authentication protocol. Finally, since the password file is w.l.o.g. an output of some one-way function applied to the password, an adaptive server

Password Authentication

- On $(\mathsf{NewSession}, sid, \mathsf{P}', \mathsf{pw}, \mathsf{role})$ from party P, send $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{role})$ to $\mathcal{A}^*$. Also, if this is the first NewSession message, or this is the second NewSession message and there is a record $\langle \mathsf{P}', \mathsf{P}, \cdot \rangle$, record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ and mark it fresh.

Active Session Attacks

- On $(\mathsf{TestPwd}, sid, \mathsf{P}, \mathsf{pw}^*)$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ marked fresh, do: if $\mathsf{pw}^* = \mathsf{pw}$, mark it compromised and return "correct guess" to $\mathcal{A}^*$; otherwise mark it interrupted and return "wrong guess."

Key Generation and Authentication

- On $(\mathsf{NewKey}, sid, \mathsf{P}, SK^*)$ from $\mathcal{A}^*$ where $|SK^*| = \kappa$, if there is a record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ not marked completed, do:
  - If the record is marked compromised, or either P or P' is corrupted, set $SK := SK^*$.
  - If the record is marked fresh, a (sid, SK') pair was sent to P', and at that time there was a record $\langle \mathsf{P}', \mathsf{P}, \mathsf{pw} \rangle$ marked fresh, set $SK := SK'$.
  - Else pick $SK \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$.

  Finally, mark $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw}' \rangle$ completed and send (sid, SK) to P.
- On $(\mathsf{NewTranscript}, sid, \mathsf{P}, \mathsf{tr}^*)$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ marked completed, do:
  - If (i) there is a record $\langle \mathsf{P}', \mathsf{P}, \mathsf{pw}' \rangle$ for which a $(\mathsf{transcript}, sid, \mathsf{tr})$ message was sent to P', and (ii) either $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ or $\langle \mathsf{P}', \mathsf{P}, \mathsf{pw}' \rangle$ was ever marked compromised or interrupted, ignore this message.
  - Else send $(\mathsf{transcript}, sid, \mathsf{tr})$ to P.

Figure 4.1: Functionality $\mathcal{F}_{\mathrm{rPAKE}}$

compromise allows the adversary to stage an offline dictionary attack: The adversary can compute the same one-way function, a.k.a. *password hash*, on any password guess, which is modeled by the OfflineTestPwd query. If the password file is stolen, this computation allows the adversary to test if its password guess is correct, because then the password hash would match the one in the password file. If the password file is not stolen yet, the adversary can store these pre-computed hashes, which $\mathcal{F}_{\text{aPAKE}}$ models by storing the password guesses made by the adversary via the OfflineTestPwd command, and learn if any of these guesses were correct at the moment of server compromise. This is modeled by functionality $\mathcal{F}_{\text{aPAKE}}$ checking after the StealPwdFile command whether any of the password guesses made via OfflineTestPwd queries is equal to the password used in to create the password file.

We note that the functionality $\mathcal{F}_{\text{aPAKE}}$ has effectively two separate notions of a server corruption. Formally, it considers a *static* adversarial model where all entities, including clients and servers, are either honest or corrupted throughout the life-time of the protocol. In addition, it allows for an *adaptive* server compromise of an honest server, via the StealPwdFile, which leaks to the adversary the server's private state corresponding to a particular password file, but it does not give the adversary full control over the server's entity. In particular, the accounts on the same server for which the adversary does not explicitly issue the StealPwdFile command must remain unaffected. We adopt this convention from [40] and we call a server "corrupted" if it is (statically) corrupted and adversarially controlled, and we call a session "compromised" if the adversary steals its password file from the server.

Non-black-box assumptions. Note that the aPAKE functionality requires the simulator, playing the role of the ideal adversary, to detect offline password guesses made by the real-world adversary. As pointed out by [40], this seems to require a non-black-box hardness assumption on some cryptographic primitive, e.g., ROM, which would allow the simulator

In the description below, we assume $\mathsf{P} \in \{\mathsf{C}, \mathsf{S}\}$.
Password Registration

- On $(\mathsf{StorePwdFile}, sid, \mathsf{C}, \mathsf{pw})$ from S, if this is the first $\mathsf{StorePwdFile}$ message, record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw} \rangle$ and mark it $\mathsf{uncompromised}$.

Stealing Password Data

- On $(\mathsf{StealPwdFile}, sid)$ from $\mathcal{A}^*$, if there is no record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw} \rangle$, return "no password file" to $\mathcal{A}^*$. Otherwise, if the record is marked $\mathsf{uncompromised}$, mark it $\mathsf{compromised}$; regardless,
  - If there is a record $(\mathsf{offline}, \mathsf{pw})$, send pw to $\mathcal{A}^*$.
  - Else return "password file stolen" to $\mathcal{A}^*$.
- On $(\mathsf{OfflineTestPwd}, sid, \mathsf{pw}^*)$ from $\mathcal{A}^*$, do:
  - If there is a record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw} \rangle$ marked $\mathsf{compromised}$, do: if $\mathsf{pw}^* = \mathsf{pw}$, return "correct guess" to $\mathcal{A}^*$; else return "wrong guess."
  - Else record $(\mathsf{offline}, \mathsf{pw})$.

Figure 4.2: Functionality $\mathcal{F}_{\text{aPAKE}}$, part 1

to extract a password guess from adversary's *local* computation, e.g., a local execution of aPAKE interaction on a password guess and a stolen password file.

Server initialization. We note that while $\mathcal{F}_{\text{aPAKE}}$ defines password registration as an internal action of the server, with the client's password as a local input, one can modify it to support an interactive procedure between client and server, e.g., to prevent the server from ever learning the plaintext password. To that end one needs to assume that during the password registration phase there is an *authenticated channel* from server to client, so the client can verify that it is registering the password with the correct server. (Functionality $\mathcal{F}_{\text{aPAKE}}$ effectively also assumes such authenticated channel because otherwise the client's password cannot be safely transported to the server.) In practice, the server also needs to verify the client's identity, and the password file could be created by the client and transported to the server. However, this is beyond the scope of the formal

Password Authentication

- On $(\mathsf{CltSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$ from C, send $(\mathsf{CltSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}^*$. Also, if this is the first CltSession message for $ssid$, record $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{pw}' \rangle$ and mark it fresh.
- On $(\mathsf{SvrSession}, sid, ssid)$ from S, retrieve $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw} \rangle$, and send $(\mathsf{SvrSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}^*$. Also, if this is the first SvrSession message for $ssid$, record $\langle ssid, \mathsf{S}, \mathsf{C}, \mathsf{pw} \rangle$ and mark it fresh.

Active Session Attacks

- On $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, \mathsf{pw}^*)$ from $\mathcal{A}^*$, if there is a record $\langle ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw}' \rangle$ marked fresh, do: if $\mathsf{pw}^* = \mathsf{pw}'$, mark it compromised and return "correct guess" to $\mathcal{A}^*$; else mark it interrupted and return "wrong guess."
- On $(\mathsf{Impersonate}, sid, ssid)$ from $\mathcal{A}^*$, if there is a record $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{pw}' \rangle$ marked fresh, do: if there is a record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw} \rangle$ marked compromised and $\mathsf{pw}' = \mathsf{pw}$, mark $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{pw}' \rangle$ compromised and return "correct guess" to $\mathcal{A}^*$; else mark it interrupted and return "wrong guess."

Key Generation and Authentication

- On $(\mathsf{NewKey}, sid, ssid, \mathsf{P}, SK^*)$ from $\mathcal{A}^*$ where $|SK^*| = \kappa$, if there is a record $\langle ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ not marked completed, do:
  - If the record is marked compromised, or either P or P' is corrupted, set $SK := SK^*$.
  - If the record is marked fresh, a $(sid, ssid, SK')$ tuple was sent to P', and at that time there was a record $\langle ssid, \mathsf{P}', \mathsf{P}, \mathsf{pw} \rangle$ marked fresh, set $SK := SK'$.
  - Else pick $SK \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$.

  Finally, mark $\langle ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ completed and send $(sid, ssid, SK)$ to P.
- On $(\mathsf{TestAbort}, sid, ssid, \mathsf{P})$ from $\mathcal{A}^*$, if there is a record $\langle ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ not marked completed, do:
  - If it is marked fresh and record $\langle ssid, \mathsf{P}', \mathsf{P}, \mathsf{pw} \rangle$ exists, send succ to $\mathcal{A}^*$.
  - Else send fail to $\mathcal{A}^*$ and $(\mathsf{abort}, sid, ssid)$ to P, and mark $\langle ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ completed.

Figure 4.3: Functionality $\mathcal{F}_{\mathrm{aPAKE}}$, part 2

aPAKE functionality.

4.3 Our PAKE-to-aPAKE Compiler

Our construction converts a symmetric UC PAKE protocol Π to an asymmetric UC PAKE, just as the compiler of Gentry *et al.* [40], but using a different method.

Our construction, shown in Figure **??** (a graphical illustration is provided in Figure 4.5), runs the symmetric PAKE protocol Π on hashed password $r = H_1(\mathsf{pw})$, but in parallel it also runs a Diffie-Hellman Key Exchange (DH-KE) where the client's contribution is fixed as $V = g^z$ for $z = H_0(\mathsf{pw})$, i.e., an independent password hash. The server's contribution, $Y = g^y$ for random y, is the only message transferred in this DH-KE instance, because the client's contribution $V = g^{H_0(\mathsf{pw})}$ is part of the password file stored on the server. The key $K_0 = V^y = Y^z = g^{H_0(\mathsf{pw})y}$ resulting from this DH-KE could be computed in an offline dictionary attack given the DH-KE transcript Y, so we hash it together with key K_1 output by the symmetric PAKE protocol Π to derive an authenticator $t = H_2(K_0||K_1|||[\ldots])$ which is sent from the client to the server before another hash of key K_1 is used as the session key. Note that security of Π implies that key K_1 is pseudorandom except if the adversary learns $r = H_1(\mathsf{pw})$ and succeeds in an *online* dictionary attack on Π, hence t is safe from *offline* dictionary attacks.

The role key K_0 plays in the derivation of authenticator t is to force the adversary to perform an offline attack against password pw *after* server compromise. Note that protocol Π plays no security role after server compromise, because the adversary can then execute Π on the correct input $r = H_1(\mathsf{pw})$. However, the DH-KE key $K_0 = Y^{H_0(\mathsf{pw})}$ is pseudorandom unless the adversary queries $H_0(\mathsf{pw})$, an event which the UC simulator (assuming ROM) can catch and identify as an offline password test. Note that the adversary who learns the server-stored

Password Registration

1. On input $(\mathsf{StorePwdFile}, sid, \mathsf{C}, \mathsf{pw})$, S computes $r := H_1(\mathsf{pw})$ and $V := g^{H_0(\mathsf{pw})}$, and records $\mathsf{file}[sid] := (r, V)$.

Password Authentication and Key Generation

1. On input $(\mathsf{CltSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$, C computes $r' := H_1(\mathsf{pw})$ and sends $(\mathsf{CltSession}, sid, ssid, \mathsf{S}, r')$ to $\mathcal{F}_{\text{rPAKE}}$.
2. On input $(\mathsf{SvrSession}, sid, ssid)$, S retrieves $\mathsf{file}[sid] = (r, V)$ and sends $(\mathsf{SvrSession}, sid, ssid, \mathsf{C}, r)$ to $\mathcal{F}_{\text{rPAKE}}$; picks $y \leftarrow_{\text{R}} \mathbb{Z}_m$ and sends $Y := g^y$ to C.
3. On $(sid, ssid, K_1')$ and $(\mathsf{transcript}, sid, ssid, \mathsf{tr}')$ from $\mathcal{F}_{\text{rPAKE}}$ and Y from S, C outputs $(\mathsf{abort}, sid, ssid)$ and halts if $Y \notin \mathbb{G}$ or $Y = 1_{\mathbb{G}}$. Else C computes $K_0' := Y^{H_0(\mathsf{pw}')}$, $t := H_2(K_0'||K_1'||Y||\mathsf{tr}')$ and $SK := H_3(K_1')$, sends t to S and outputs $(sid, ssid, SK)$.
4. On $(sid, ssid, K_1)$ and $(\mathsf{transcript}, sid, ssid, \mathsf{tr})$ from $\mathcal{F}_{\text{rPAKE}}$ and t from C, S computes $K_0 := V^y$ and outputs $(\mathsf{abort}, sid, ssid)$ and halts if $t \neq H_2(K_0||K_1||Y||\mathsf{tr})$. Else S computes $SK := H_3(K_1)$ and outputs $(sid, ssid, SK)$.

Figure 4.4: Compiler from symmetric PAKE to asymmetric PAKE in the $\mathcal{F}_{\text{rPAKE}}$-hybrid world

values $r = H_1(\mathsf{pw})$ and $V = g^{H_0(\mathsf{pw})}$ can also perform an offline test by hashing its password guesses via H_0 and H_1, but the point is that our DH-KE instance key K_1 does not offer any *easier* way for the adversary to find a password than an offline dictionary attack, which is unavoidable in the asymmetric PAKE setting after server compromise.

We state the security of our aPAKE protocol in Theorem 6 below:

Theorem 6. *Suppose that the CDH assumption holds for $(\mathbb{G}, g, m)$. Then the protocol in Figure 4.4 realizes functionality $\mathcal{F}_{\text{aPAKE}}$ in ROM.*

Proof. For any efficient environment $\mathcal{Z}$ and efficient adversary $\mathcal{A}$ against the protocol, we construct a simulator SIM as in Figure 4.6, Figure 4.7 and Figure 4.8. Without loss of generality, suppose $\mathcal{A}$ is a "dummy" adversary who merely passes through all its messages to and from $\mathcal{Z}$, and all its computation to $\mathcal{Z}$. To keep notation brief we denote functionality

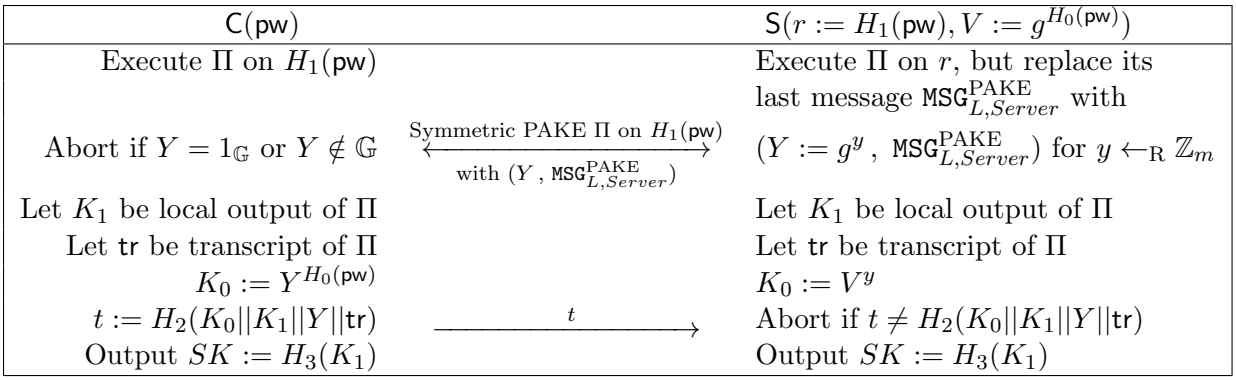

Figure 4.5: Compiler from symmetric PAKE to asymmetric PAKE (graphical illustration)

$\mathcal{F}_{\text{aPAKE}}$ as $\mathcal{F}$.

We now show that the distinguishing advantage of $\mathcal{Z}$ between the real world and the simulated world is negligible. As before, the argument uses a sequence of games, starting from the real world and ending at the simulated world.

$\mathbf{G}_0$ is the real world.

In $\mathbf{G}_1$, in the case that S is compromised, on $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, r^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{rPAKE}}$ where $r^* = r$, return "correct guess" if $\mathsf{pw}' = \mathsf{pw}$, and "wrong guess" otherwise.

In $\mathbf{G}_0$, $\mathcal{A}$ receives "correct guess" if and only if $r^* = r = H_1(\mathsf{pw}')$. By definition $r = H_1(\mathsf{pw})$. $\mathcal{Z}$'s views in $\mathbf{G}_0$ and $\mathbf{G}_1$ are identical unless $\mathsf{pw}' \neq \mathsf{pw}$ but $H_1(\mathsf{pw}') = H_1(\mathsf{pw}) = r$ (in which case $\mathcal{A}$ receives "correct guess" in $\mathbf{G}_0$ and "wrong guess" in $\mathbf{G}_1$). Since both $H_1(\mathsf{pw})$ and $H_1(\mathsf{pw}')$ are random strings in $\{0,1\}^\kappa$, for a single C sub-session, the probability of the above is $1/2^\kappa$; assuming that there are q_{C} C sub-sessions, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_0,\mathbf{G}_1} \leq \frac{q_{\mathrm{C}}}{2^\kappa},$$

which is a negligible function of κ.

In $\mathbf{G}_2$, in the case that S is not compromised, on $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, r^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{rPAKE}}$,

Stealing Password Data and Offline Queries

1. On $(\mathsf{StealPwdFile}, sid)$ from $\mathcal{A}$ aimed at S, pass this message to $\mathcal{F}$.
   If $\mathcal{F}$ returns "no password file," pass this message to $\mathcal{A}$ as a message from S.
   If $\mathcal{F}$ returns "password file stolen," pick $r \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$ and $v \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, and record $(\perp, r, v)$.
   If $\mathcal{F}$ returns pw, record (pw, r, v) where $r := H_1(\mathsf{pw})$ and $v := H_0(\mathsf{pw})$ (pick $H_1(\mathsf{pw})$ and/or $H_0(\mathsf{pw})$ at random if undefined); if there is a session record $\langle ssid, \mathsf{S}, \mathsf{C}, \perp, \perp, \cdot, \cdot \rangle$, then also modify it to $\langle ssid, \mathsf{S}, \mathsf{C}, \mathsf{pw}, r, \cdot, \cdot \rangle$.

2. On $\mathcal{A}$ making a fresh query $H_1(x)$ or $H_0(x)$, if there is a record (pw, r, v) and $x = \mathsf{pw}$ (i.e., $\mathcal{A}$ has made one of these two queries [note that when (pw, r, v) is recorded, $\mathcal{A}$ must have queried either $H_1(\mathsf{pw})$ or $H_0(\mathsf{pw})$] and is querying the other), send r (if $\mathcal{A}$ queries $H_1(\mathsf{pw})$) or v (if $\mathcal{A}$ queries $H_0(\mathsf{pw})$) to $\mathcal{A}$.
   Else send $(\mathsf{OfflineTestPwd}, sid, x)$ to $\mathcal{F}$.
   If there is no response from $\mathcal{F}$, or $\mathcal{F}$ returns "wrong guess," then pick $r \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$ and $v \leftarrow_{\mathrm{R}} \mathbb{Z}_m$.
   If $\mathcal{F}$ returns "correct guess", retrieve $(\perp, r, v)$ (note that in this case S is compromised, so there is such a record) and replace $\perp$ with x.
   In either case, set $H_1(x) := r$ and $H_0(x) := v$, and send r (if $\mathcal{A}$ queries $H_1(x)$) or v (if $\mathcal{A}$ queries $H_0(x)$) to $\mathcal{A}$.

Figure 4.6: The simulator SIM for aPAKE in the stealing password data phase

if there is a record $\langle ssid, \mathsf{P}, \mathsf{P}', \cdot \rangle$ marked fresh, check if there is an x such that $r^* = H_1(x)$.

- If there are more than one such x's, output $\mathsf{collision}$ and abort.
- If there is a unique such x and $x = \mathsf{pw}'$ (if $\mathsf{P} = \mathsf{C}$) or $x = \mathsf{pw}$ (if $\mathsf{P} = \mathsf{S}$), send "correct guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathrm{rPAKE}}$.
- In all other cases (i.e., $x \neq \mathsf{pw}'/\mathsf{pw}$ or there is no such x), send "wrong guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathrm{rPAKE}}$.

First consider event $\mathsf{collision}$. $\mathsf{collision}$ occurs if and only if there are more than one x's such that $r^* = H_1(x)$. This means that there are $x_1 \neq x_2$ such that $H_1(x_1) = H_1(x_2)$. Assuming that $\mathcal{A}$ queries $H_1(\cdot)$ q_{H1} times, there are at most $q_{\mathrm{H1}} + q_{\mathrm{C}} + 1$ $H_1(\cdot)$ queries in total (q_{H1}

Password Authentication

1. On $(\mathsf{CltSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}$, send $(\mathsf{NewSession}, sid, ssid, \mathsf{C}, \mathsf{S}, \mathsf{client})$ to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{rPAKE}}$. Also, if this is the first $\mathsf{CltSession}$ message for $ssid$, record the session record for C $\langle ssid, \mathsf{C}, \mathsf{S}, \perp, \perp, \perp, \perp\rangle$ and mark it fresh.

2. On $(\mathsf{SvrSession}, sid, ssid, \mathsf{S}, \mathsf{C})$ from $\mathcal{F}$, send $(\mathsf{NewSession}, sid, ssid, \mathsf{S}, \mathsf{C}, \mathsf{server})$ to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{rPAKE}}$. Also, if this is the first $\mathsf{SvrSession}$ message for $ssid$, record the session record for S $\langle ssid, \mathsf{S}, \mathsf{C}, \mathsf{pw}, r, \perp, \perp\rangle$ (if there is a record $(\mathsf{pw}, r, \cdot)$) or $\langle ssid, \mathsf{S}, \mathsf{C}, \perp, \perp, \perp, \perp\rangle$ (if there is no such record) and mark it fresh. Furthermore, pick $y \leftarrow_{\text{R}} \mathbb{Z}_m$ and record $(ssid, y, \perp, \perp)$; compute $Y := g^y$ and send Y to $\mathcal{A}$ as a message from S to C.

Protocol Messages in PAKE

1. On $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, r^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{rPAKE}}$, if there is P's session record marked fresh, do:

   - If there is a record $(\cdot, r, \cdot)$ (i.e., S is compromised), then: (1) if $\mathsf{P} = \mathsf{S}$: if $r^* = r$, then return "correct guess" to $\mathcal{A}$; else return "wrong guess." (2) if $\mathsf{P} = \mathsf{C}$: if $r^* = r$, then send $(\mathsf{Impersonate}, sid, ssid)$ to $\mathcal{F}$, and pass $\mathcal{F}$'s response to $\mathcal{A}$; if $\mathcal{F}$ returns "correct guess," modify C's session record to $\langle ssid, \mathsf{S}, \mathsf{C}, \cdot, r, \perp, \perp\rangle$ and mark this case $(*)$. Else move on to the next case.
   - Else check if there is an x such that $r^* = H_1(x)$.
     - If there are more than one such x's, output $\mathsf{collision}$ and abort.
     - If there is a unique such x, send $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, x)$ to $\mathcal{F}$ and pass $\mathcal{F}$'s response to $\mathcal{A}$.
       If $\mathcal{F}$ returns "correct guess," then modify P's session record to $\langle ssid, \mathsf{P}, \mathsf{P}', x, r^*, \perp, \perp\rangle$. Furthermore, if $\mathsf{P} = \mathsf{C}$, mark this case $(**)$; if $\mathsf{P} = \mathsf{S}$, record (x, r^*, v) where $v = H_0(x)$ (pick $v := H_0(x) \leftarrow_{\text{R}} \mathbb{Z}_m$ if undefined).
     - If there is no such x, return "wrong guess" to $\mathcal{A}$.

   Finally, if the message is "correct guess," mark P's session record $\mathsf{compromised}$; if the message is "wrong guess," mark it $\mathsf{interrupted}$.

2. On $(\mathsf{NewKey}, sid, ssid, \mathsf{P}, K_1^*)$ (where $|K_1^*| = \kappa$) from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{rPAKE}}$, if there is P's session record not marked $\mathsf{completed}$, mark it $\mathsf{completed}$. Furthermore, if P's session record is marked $\mathsf{compromised}$, or either P or P' is corrupted, then set $K_1 := K_1^*$ and modify the session record to $\langle ssid, \mathsf{P}, \mathsf{P}', \cdot, \cdot, K_1, \perp\rangle$.

3. On $(\mathsf{NewTranscript}, sid, ssid, \mathsf{P}, \mathsf{tr})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{rPAKE}}$, ignore this message if there is no session record $\langle ssid, \mathsf{P}, \mathsf{P}', \cdot, \cdot, \cdot, \perp\rangle$ marked $\mathsf{completed}$; or if (i) there is a peer session record for P' $\langle ssid, \mathsf{P}', \mathsf{P}, \cdot, \cdot, \cdot, \mathsf{tr}\rangle$, and (ii) either of these two records was ever marked $\mathsf{compromised}$ or $\mathsf{interrupted}$. Else modify P's session record to $\langle ssid, \mathsf{P}, \mathsf{P}', \cdot, \cdot, \cdot, \mathsf{tr}\rangle$.

Figure 4.7: The simulator SIM for aPAKE in the PAKE protocol

Protocol Messages

1. On Y^* from $\mathcal{A}$ aimed at C, if $Y^* = 1_{\mathbb{G}}$ or $Y^* \notin \mathbb{G}$, send $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, \perp)$ and then $(\mathsf{TestAbort}, sid, ssid, \mathsf{C})$ to $\mathcal{F}$.
   Else if there is C's session record whose seventh item is a string tr' and a record $(ssid, y, \perp, \perp)$, modify the latter to $(ssid, y, Y^*, t)$; once C's session record is marked $\mathsf{completed}$, send t to $\mathcal{A}$ (as a message from C to S) and $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, SK')$ to $\mathcal{F}$, where t and SK' are defined as follows:

   - If the sixth item of the session record is a string K_1' (this means that the record was marked $\mathsf{compromised}$, which in turn means that $\mathcal{A}$ has sent $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, r' = H_1(\mathsf{pw}'))$ aimed at $\mathcal{F}_{\text{rPAKE}}$), and the fourth item of the session record is $\perp$ (this means that case $(*)$ happens, which in turn means that S is compromised and $\mathsf{pw}' = \mathsf{pw}$), then retrieve $(\cdot, r, v)$ and compute $K_0' := (Y^*)^v$; pick $t \leftarrow_{\text{R}} \{0,1\}^\kappa$ and set $H_2(K_1'||K_0'||Y^*||\mathsf{tr}') := t$; pick $H_3(K_1') := SK' \leftarrow_{\text{R}} \{0,1\}^\kappa$.
   - If the sixth item of the session record is a string K_1', the fourth item of the session record is a password pw' (this means that case $(**)$ happens, which in turn means that $H_1(\mathsf{pw}')$ has been queried), do:
     - If both $H_0(\mathsf{pw}')$ and $H_2(K_1'||K_0|'|Y^*||\mathsf{tr}')$ where $K_0' = (Y^*)^{H_0(\mathsf{pw}')}$ have been queried, then set $t := H_2(K_1'||K_0'||Y^*||\mathsf{tr}')$ and pick $H_3(K_1') := SK' \leftarrow_{\text{R}} \{0,1\}^\kappa$.
     - Else pick $t \leftarrow_{\text{R}} \{0,1\}^\kappa$ and $H_3(K_1') := SK' \leftarrow_{\text{R}} \{0,1\}^\kappa$. Furthermore, on $\mathcal{A}$ querying $H_0(\mathsf{pw}')$ and $H_2(K_1'||K_0'||Y^*||\mathsf{tr}')$ where $K_0' = (Y^*)^{H_0(\mathsf{pw}')}$, set $H_2(K_1'||K_0'||Y^*||\mathsf{tr}') := t$.
   - Else, i.e., if the sixth item of the session record is $\perp$, pick $t \leftarrow_{\text{R}} \{0,1\}^\kappa$ and set $SK' := 0^\kappa$.

2. On t^* from $\mathcal{A}$ aimed at S, if S's session record is marked $\mathsf{completed}$ and its seventh item is a string tr, and there is a record $(ssid, y, \cdot, \cdot)$, do:

   - If the record is $(ssid, y, Y^*, t)$, $Y^* = g^y$ and $t^* = t$, there is C's session record whose seventh item is tr, then send $(\mathsf{TestAbort}, sid, ssid, \mathsf{S})$ to $\mathcal{F}$. If $\mathcal{F}$ returns succ, send $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, 0^\kappa)$ to $\mathcal{F}$.
   - Else if there are records $\langle ssid, \mathsf{S}, \mathsf{C}, \mathsf{pw}, r, K_1, \mathsf{tr}\rangle$ and (pw, r, v), and $t^* = H_2(K_1||K_0||Y||\mathsf{tr})$ where $K_0 = g^{vy}$, then pick $H_3(K_1) := SK \leftarrow_{\text{R}} \{0,1\}^\kappa$ and send $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, \mathsf{pw})$ and then $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, SK)$ to $\mathcal{F}$.
   - Else send $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, \perp)$ and then $(\mathsf{TestAbort}, sid, ssid, \mathsf{S})$ to $\mathcal{F}$.

Random Oracle Queries

Note that how to answer H_1 and H_0 queries has been described in "Stealing Password Data." Answer H_2 and H_3 queries via lazy sampling, except in the cases described in "Protocol Messages."

Figure 4.8: The simulator SIM for aPAKE in the login phase

queries by $\mathcal{A}$, q_{C} by C and 1 by S), so we have that

$$\Pr[\mathsf{collision}] \leq \frac{(q_{\mathrm{H1}} + q_{\mathrm{C}} + 1)^2}{2^{\kappa+1}}.$$

Next assume that $\mathsf{collision}$ does not occur. Consider the first message of type $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, r^*)$ (note that $\mathcal{A}$ receives a reply for the first such message only, since $\langle ssid, \mathsf{P}, \mathsf{P}', \cdot \rangle$ becomes either $\mathsf{compromised}$ or $\mathsf{interrupted}$ after the first message). In both $\mathbf{G}_1$ and $\mathbf{G}_2$, $\mathcal{A}$ receives "correct guess" if and only if $r^* = H_1(\mathsf{pw}')$ (if $\mathsf{P} = \mathsf{C}$) or $r^* = H_1(\mathsf{pw})$ (if $\mathsf{P} = \mathsf{S}$), so $\mathcal{Z}$'s views in $\mathbf{G}_1$ and $\mathbf{G}_2$ in this case are identical. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_1,\mathbf{G}_2} \leq \Pr[\mathsf{collision}] \leq \frac{(q_{\mathrm{H1}} + q_{\mathrm{C}} + 1)^2}{2^{\kappa+1}}.$$

which is a negligible function of κ.

$\mathbf{G}_3$ considers the case that $\mathcal{A}$ queries $H_1(\mathsf{pw}')$ (denote the result r'), sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, r')$ to $\mathcal{F}_{\mathrm{rPAKE}}$, and then sends Y^* to C. If $\mathcal{A}$ additionally queries both $H_0(\mathsf{pw}')$ and $H_2(K_1'||K_0'||Y^*||\mathsf{tr}')$ (denote the result t) where $K_0' = (Y^*)^{H_0(\mathsf{pw}')}$ (call these two queries "crucial queries"), then send t to $\mathcal{A}$ as a message from C to S, just as in $\mathbf{G}_2$. Otherwise send $t \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$ to $\mathcal{A}$ as a message from C to S, and when $\mathcal{A}$ makes the "crucial queries," set $H_2(K_1'||K_0'||Y^*||\mathsf{tr}') := t$. Clearly this modification does not change the distribution of t in $\mathcal{Z}$'s view, so we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_2,\mathbf{G}_3} = 0.$$

In $\mathbf{G}_4$, if $\mathcal{A}$ does not send $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, r')$ to $\mathcal{F}_{\mathrm{rPAKE}}$ (where $r' = H_1(\mathsf{pw}')$), send $t \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$ to $\mathcal{A}$ as a message from C to S, and C outputs $SK' \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$.

In $\mathbf{G}_3$, t (resp. SK') is a random string in $\{0,1\}^\kappa$ in $\mathcal{Z}$'s view unless and until $\mathcal{A}$ queries

$H_2(K_1'||K_0'||Y^*||\mathsf{tr}')$ (resp. $H_3(K_1')$). But C's session in $\mathcal{F}_{\text{rPAKE}}$[1] is never marked $\mathsf{compromised}$, so C's PAKE output K_1' is a random string in $\{0,1\}^\kappa$ in $\mathcal{Z}$'s view. Assuming that $\mathcal{A}$ queries $H_2(\cdot)$ q_{H2} times and $H_3(\cdot)$ q_{H3} times, the probability that $\mathcal{A}$ makes such a query for a single K_1' is at most $(q_{\text{H2}} + q_{\text{H3}})/2^\kappa$. Since there are q_{C} K_1''s, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_3,\mathbf{G}_4} \leq \frac{q_{\text{C}}(q_{\text{H2}} + q_{\text{H3}})}{2^\kappa},$$

which is a negligible function of κ.

In $\mathbf{G}_5$, on t^* from $\mathcal{A}$ to S (after S sends Y to C [relayed by $\mathcal{A}$ as Y^*]), if $Y^* = Y$, $t^* = t$, and $\mathsf{tr}' = \mathsf{tr}$, do:

- If $\mathsf{pw}' = \mathsf{pw}$, then S picks $SK \leftarrow_{\text{R}} \{0,1\}^\kappa$ and outputs $(sid, ssid, SK)$;
- If $\mathsf{pw}' \neq \mathsf{pw}$, then S outputs $(\mathsf{abort}, sid, ssid)$.

This is the case that $\mathcal{A}$ merely passes all messages between C and S (note that condition $\mathsf{tr}' = \mathsf{tr}$ implies that both C and S's sessions in $\mathcal{F}_{\text{rPAKE}}$ were never marked $\mathsf{compromised}$ or $\mathsf{interrupted}$, i.e., $\mathcal{A}$ passes all messages between C and S in the PAKE protocol). First consider the subcase that $\mathsf{pw}' = \mathsf{pw}$. In $\mathbf{G}_4$, SK is a random string in $\{0,1\}^\kappa$ in $\mathcal{Z}$'s view unless and until $\mathcal{A}$ queries $H_3(K_1)$. But S's session in $\mathcal{F}_{\text{rPAKE}}$ was never marked $\mathsf{compromised}$, so S's PAKE output K_1 is a random string in $\{0,1\}^\kappa$ in $\mathcal{Z}$'s view, thus the probability that $\mathcal{A}$ makes such a query for a single K_1 is at most $q_{\text{H3}}/2^\kappa$. Assuming that there are q_{S} S sub-sessions, there are q_{S} K_1's, so $\mathcal{Z}$'s distinguishing advantage in the subcase that $\mathsf{pw}' = \mathsf{pw}$ is at most $q_{\text{S}}q_{\text{H3}}/2^\kappa$.

Next consider the subcase that $\mathsf{pw}' \neq \mathsf{pw}$. $\mathcal{Z}$'s views in $\mathbf{G}_4$ and $\mathbf{G}_5$ in this subcase are identical unless $\mathsf{pw}' \neq \mathsf{pw}$ but $t^* = t = H_2(K_1||K_0||Y||\mathsf{tr})$ (in which case S outputs $(sid, ssid, SK)$ in

[1]Note that a C (or S) session in $\mathcal{F}_{\text{rPAKE}}$ corresponds to a C sub-session in the aPAKE protocol, with $(sid, ssid)$ as the session ID.

$\mathbf{G}_4$ and $(\mathsf{abort}, sid, ssid)$ in $\mathbf{G}_5$). By definition $t = H_2(K_1'||K_0'||Y^*||\mathsf{tr}') = H_2(K_1||K_0'||Y||\mathsf{tr})$, $K_0 = g^{H_0(\mathsf{pw})y}$, and $K_0' = g^{H_0(\mathsf{pw}')y}$. Therefore, if $t = H_2(K_1||K_0||Y||\mathsf{tr})$, either $H_0(\mathsf{pw}') = H_0(\mathsf{pw})$, or $H_0(\mathsf{pw}') \neq H_0(\mathsf{pw})$ but $H_2(K_1||K_0'||Y||\mathsf{tr}) = H_2(K_1||K_0||Y||\mathsf{tr})$. Since there are q_C pw''s and q_C K_0''s, the probability of the above is at most $q_\mathrm{C}/2^\kappa + q_\mathrm{C}/2^\kappa = q_\mathrm{C}/2^{\kappa-1}$.

Combining the two results above, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_4,\mathbf{G}_5} \leq \frac{2q_\mathrm{C} + q_\mathrm{S}q_{\mathrm{H3}}}{2^\kappa},$$

which is a negligible function of κ.

In $\mathbf{G}_6$, on t^* from $\mathcal{A}$ to S (after S sends Y to C), in the case that $\neg(Y^* = Y \wedge t^* = t \wedge \mathsf{tr}' = \mathsf{tr})$, S outputs $(\mathsf{abort}, sid, ssid)$ if $\mathcal{A}$ did not query $H_2(K_1||K_0||Y||\mathsf{tr})$.

$\mathcal{Z}$'s views in $\mathbf{G}_5$ and $\mathbf{G}_6$ are identical unless $\mathcal{A}$ does not query $H_2(K_1||K_0||Y||\mathsf{tr})$ but $t^* = H_2(K_1||K_0||Y||\mathsf{tr})$ (in which case S outputs $(sid, ssid, SK)$ in $\mathbf{G}_5$ and $(\mathsf{abort}, sid, ssid)$ in $\mathbf{G}_6$). In this case $H_2(K_1||K_0||Y||\mathsf{tr})$ is a random string in $\{0,1\}^\kappa$ in $\mathcal{Z}$'s view, so $\Pr[t^* = H_2(K_1||K_0||Y||\mathsf{tr})] \leq 1/2^\kappa$ for a single t^*. Since there are q_S t^*'s, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_5,\mathbf{G}_6} \leq \frac{q_\mathrm{S}}{2^\kappa},$$

which is a negligible function of κ.

In $\mathbf{G}_7$, on t^* from $\mathcal{A}$ to S (after S sends Y to C), in the case that $\neg(Y^* = Y \wedge t^* = t \wedge \mathsf{tr}' = \mathsf{tr})$, S outputs $(\mathsf{abort}, sid, ssid)$ if $\mathcal{A}$ did not send $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, r)$ to $\mathcal{F}_{\mathrm{rPAKE}}$.

$\mathcal{Z}$'s views in $\mathbf{G}_6$ and $\mathbf{G}_7$ are identical unless $\mathcal{A}$ does not send $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, r)$ to $\mathcal{F}_{\mathrm{rPAKE}}$ but queries $H_2(K_1||K_0||Y||\mathsf{tr})$ (in which case S outputs $(sid, ssid, SK)$ in $\mathbf{G}_6$ and $(\mathsf{abort}, sid, ssid)$ in $\mathbf{G}_7$). But S's session in $\mathcal{F}_{\mathrm{rPAKE}}$ is never marked $\mathsf{compromised}$, so S's PAKE output K_1 is a random string in $\{0,1\}^\kappa$ in $\mathcal{Z}$'s view, thus the probability that $\mathcal{A}$

makes such a query for a single K_1 is at most $q_{\mathrm{H2}}/2^{\kappa}$. Since there are q_{S} K_1's, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_6,\mathbf{G}_7} \leq \frac{q_{\mathrm{S}} q_{\mathrm{H2}}}{2^{\kappa}},$$

which is a negligible function of κ.

In $\mathbf{G}_8$, on t^* from $\mathcal{A}$ to S (after S sends Y to C), in the case that $\neg(Y^* = Y \wedge t^* = t \wedge \mathsf{tr}' = \mathsf{tr})$, S outputs $(\mathsf{abort}, sid, ssid)$ if $\mathcal{A}$ did not send $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, r)$ to $\mathcal{F}_{\mathrm{rPAKE}}$ (as in $\mathbf{G}_7$) or query $H_0(\mathsf{pw})$.

$\mathcal{Z}$'s views in $\mathbf{G}_7$ and $\mathbf{G}_8$ are identical unless $\mathcal{A}$ does not query $H_0(\mathsf{pw})$ but queries $H_2(K_1||K_0||Y||\mathsf{tr})$ (in which case S outputs $(sid, ssid, SK)$ in $\mathbf{G}_7$ and $(\mathsf{abort}, sid, ssid)$ in $\mathbf{G}_8$). Denote this event bad. In this event what $\mathcal{Z}$ learns contains $V = g^{H_0(\pi)}$ and $Y = g^y$ only, and according to the CDH assumption in $(\mathbb{G}, g, m)$, $\mathcal{Z}$ cannot compute $K_0 = g^{H_0(\pi)y}$ except with negligible probability. Concretely, we construct a reduction $\mathcal{R}$ to the CDH problem in $(\mathbb{G}, g, m)$:

$\mathcal{R}$, on inputs X and Y, picks $i \leftarrow_{\mathrm{R}} \{1, \ldots, q_{\mathrm{S}}\}$ (a guess of bad occurs at which S sub-session) and $j \leftarrow_{\mathrm{R}} \{1, \ldots, q_{\mathrm{H2}}\}$ (a guess of $\mathcal{A}$ causes bad at which $H_2(\cdot)$ query). Then $\mathcal{R}$ runs the challenger of $\mathbf{G}_7$, except that (1) on $(\mathsf{StealPwdFile}, sid, \mathsf{S})$ from $\mathcal{A}$ aimed at S, $\mathcal{R}$ uses X in the CDH challenge (instead of $g^{H_0(\mathsf{pw})}$) as the response, and if $\mathcal{A}$ queries $H_0(\mathsf{pw})$ before the i-th C sub-session or before making the j-th $H_2(\cdot)$ query, $\mathcal{R}$ aborts; (2) in the i-th S sub-session, on $(\mathsf{SvrSession}, sid, ssid, \mathsf{S}, \mathsf{C})$ from $\mathcal{Z}$, $\mathcal{R}$ uses Y in the CDH challenge (instead of g^y for $y \leftarrow_{\mathrm{R}} \mathbb{Z}_m$) as the message from S to C. When $\mathcal{A}$ makes the j-th query to $H_2(\cdot)$, $\mathcal{R}$ parses $\mathcal{A}$'s input as $[K_1||Z||Y||\mathsf{tr}]$, and outputs Z as the solution to the CDH problem.

We can see that if bad occurs and $\mathcal{R}$'s guesses i and j are both correct, then $\mathcal{R}$ solves the

CDH problem. Therefore, we have that

$$\mathbf{Adv}_{\mathcal{R}}^{\mathrm{CDH},\mathbb{G}} \geq \frac{1}{q_{\mathrm{S}} q_{\mathrm{H2}}} \cdot \Pr[\mathsf{bad}],$$

so

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_7,\mathbf{G}_8} \leq \Pr[\mathsf{bad}] \leq q_{\mathrm{S}} q_{\mathrm{H2}} \cdot \mathbf{Adv}_{\mathcal{R}}^{\mathrm{CDH},\mathbb{G}},$$

which is a negligible function of κ.

In $\mathbf{G}_9$, at the beginning of the game, pick $r \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$ (instead of $r := H_1(\mathsf{pw})$) and $v \leftarrow_{\mathrm{R}} \mathbb{Z}_m$ (instead of $v := H_0(\mathsf{pw})$), set $V := g^v$, and leave $H_1(\mathsf{pw})$ and $H_0(\mathsf{pw})$ undefined. In addition, if $\mathcal{A}$ sends $(\mathsf{StealPwdFile}, sid)$ to S and then queries $H_1(\mathsf{pw})$ (resp. $H_0(\mathsf{pw})$), set $H_1(\mathsf{pw}) := r$ (resp. $H_0(\mathsf{pw}) := v$). Clearly this modification does not change the distribution of r and V in $\mathcal{Z}$'s view, so we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_8,\mathbf{G}_9} = 0.$$

Note that in $\mathbf{G}_9$, $r' = H_1(\mathsf{pw}')$ is defined no matter whether $\mathcal{A}$ queries it or not. In $\mathbf{G}_{10}$, leave r' undefined unless and until $\mathcal{A}$ queries $H_1(\mathsf{pw}')$.

If $\mathcal{A}$ does not query $H_1(\mathsf{pw}')$, then r' is a random string in $\{0,1\}^\kappa$ in $\mathcal{Z}$'s view. Since there are q_{C} r''s, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_9,\mathbf{G}_{10}} \leq \frac{q_{\mathrm{C}}}{2^\kappa},$$

which is a negligible function of κ.

$\mathbf{G}_{11}$ is the simulated world. We can see that the change from $\mathbf{G}_{10}$ to $\mathbf{G}_{11}$ is merely conceptual, with the game challenger split into the aPAKE functionality $\mathcal{F}$ and the simulator SIM. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_{10},\mathbf{G}_{11}} = 0.$$

Summing up all results above, we conclude that $\mathcal{Z}$'s distinguishing advantage between the real world and the simulated world is a negligible function of κ. This completes the proof. $\square$

Chapter 5

OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks

In this chapter we initiate the study of *strong aPAKE (saPAKE)* protocols that strengthen the aPAKE security notion by *disallowing pre-computation attacks*, hence forcing the adversary to perform full offline dictionary attack *upon compromising the server* (which takes $O(|D|)$ time where D is the password dictionary). Our contribution is three-fold:

- *Security model:* We formalize the saPAKE notion in the UC framework by modifying the aPAKE functionality from [40] to eliminate an adversarial action which allowed pre-computation attacks. As explained in Chapter 1, allowing pre-computation attacks was indeed necessary to model the security of existing aPAKE protocols.
- *saPAKE protocols:* We present two generic constructions. The first builds the saPAKE protocol from any aPAKE protocol (namely one that satisfies the original definition from [40]) so that one can "salvage" existing aPAKE protocols. To do so we resort

to Oblivious PRF (OPRF), as introduced in Chapter 3. We show that by preceding any aPAKE protocol with an OPRF interaction in which the client computes the value $\mathsf{rw} = F_k(\mathsf{pw})$ (rw for "randomized password") with the help of the server and uses rw as the password in the aPAKE protocol, one obtains a strong aPAKE protocol. We show that if the OPRF and the given aPAKE protocol are, respectively, UC realizations of the OPRF functionality (defined in Section 3.2, with some minor revisions detailed below) and the original aPAKE functionality from [40], the resultant protocol realizes our UC functionality $\mathcal{F}_{\text{saPAKE}}$.

Our second transformation consists of the composition of an OPRF as above with a regular Authenticated Key Exchange (AKE) protocol, with the server being the last-to-complete party.[1] We require UC security for the AKE protocol as well as a property known as resistance to Key-Compromise Impersonation (KCI) attacks. The latter means that an attacker that learns the secret keys of one party P, but does not actively control P, cannot use this information to impersonate another party P' to P. KCI resistance is a common property of most AKE protocols. In our saPAKE construction, the client first runs the OPRF with the server to compute $\mathsf{rw} = F_k(\mathsf{pw})$, as in the first construction; then it runs the AKE protocol with the server using a private key stored, encrypted using an *authenticated* encryption under rw, at the server who sends it to the client. Crucial to the security of the protocol is the use of authenticated encryption with a "random-key robustness" property, which is achieved naturally by some schemes or otherwise can be easily ensured, e.g., by adding an HMAC to a symmetric encryption scheme. Under these conditions we show that the composed protocol realizes our UC functionality $\mathcal{F}_{\text{saPAKE}}$.

- *Concrete instantiation:* We use the above second transformation to instantiate an saPAKE protocol with a very efficient OPRF and any efficient AKE with the KCI

[1] We also show that if the client is the last-to-complete party, then the composition realizes a relaxed saPAKE functionality; see Section **??** for more details.

property. The OPRF protocol we use, 2HashDH, is proven UC secure in Section 3.2; however, we extend the 2HashDH protocol and show that it remains secure in spite of changes to the OPRF functionality that we introduce for supporting a stronger OPRF notion needed in our setting. We call the result of this instantiation, the *OPAQUE protocol.*

The results presented in this chapter are based on the work of [46], with significant revision of the security proofs. [46] incorrectly claims that the 2-round OPAQUE protocol realizes the $\mathcal{F}_{\text{saPAKE}}$ functionality; we correct this mistake by adding a round of explicit authentication, yielding a 3-round protocol.

5.1 Overview

Properties of OPAQUE. OPAQUE combines the best properties of existing aPAKE protocols and of the standard password-over-TLS approach of password authentication. As any aPAKE-secure protocol, it offers two fundamental advantages over the TLS-based solution: It does not rely on PKI and the plaintext password is never in the clear at the server. The only way for an attacker that observes (or actively controls) a session at a server to learn the password is via an exhaustive offline dictionary attack. Watching or participating in a session with the client does not help the attacker. At the same time, OPAQUE resolves the major flaw of existing aPAKE protocols relative to password-over-TLS, namely, their vulnerability to pre-computation attacks.

In addition to the above fundamental properties, OPAQUE enjoys important properties for use in practice. Its modularity allows for its use with different AKE protocols that can provide different features and performance tradeoffs. When implemented with a 3-message AKE protocol with explicit authentication (e.g., HMQV-C [56]), OPAQUE takes only 3

messages. The computational cost (using the 2HashDH protocol from Section 3.2) is 1 exponentiation for the server and 2 for the client in addition to the AKE protocol cost (with HMQV, this cost is 2.17 exponentiations per party). OPAQUE offers forward secrecy (a particularly crucial property for password protocols) if the AKE does.

OPAQUE further supports password hardening for increasing the cost of offline dictionary attacks (upon server compromise) through client-side iterated hashing without the need to transmit salt from the server to the client. In Figure 5.11 in Section 5.6 we show an instantiation of OPAQUE in ROM with HMQV as the AKE.

Compared to the practical aPAKE protocols that have been and are being considered for standardization (cf., [1, 68]), OPAQUE fares clearly better on the security side as the only protocol that offers resistance to pre-computation attacks. Performance-wise, OPAQUE is competitive with the more efficient among these protocols (see Section 5.6).

OPAQUE also provides a unique functionality among aPAKE protocols in that it allows to *store and retrieve the client's secrets* such as a bitcoin wallet, authentication credentials, encrypted backup keys, etc., thus offering a far more secure alternative to the practice of deriving low-entropy secrets directly from a client's password. Furthermore, OPAQUE allows for a client-transparent server-side threshold implementation (as detailed in Chapter 3) where the only exposure of the client's password – or any stored secrets – is in case a threshold of servers is compromised and even then a full dictionary attack is required.

Finally, we comment that while OPAQUE can completely replace password authentication in TLS, it can also be used in conjunction with TLS for protecting account information, for bootstrapping TLS client authentication (via an OPAQUE-retrieved client signing key), or as an hedge against PKI failures. In other words, while we are accustomed to use TLS to protect passwords, OPAQUE can be used to protect TLS. We expand on this aspect in Section 5.6.2.

Related work. We note that variants of OPAQUE have been studied in prior work in several settings. While our treatment frames OPAQUE in the context of OPRF [43], its design can be seen as an instantiation of the Ford-Kaliski paradigm for password hardening and credential retrieval using Chaum's blinded exponentiation. Boyen [18] specifies and studies the protocol (called HPAKE) in the setting of client-side *halting KDF* [17]. However, none of these works presents a formal analysis of the protocol as an aPAKE, let alone as a strong aPAKE, a notion that we introduce here for the first time.

5.2 The Strong aPAKE Functionality $\mathcal{F}_{\text{saPAKE}}$

We present the ideal UC strong aPAKE functionality, $\mathcal{F}_{\text{saPAKE}}$, that will serve as our definition of strong aPAKE security; namely, we call a protocol a secure *strong aPAKE (saPAKE)* if it realizes $\mathcal{F}_{\text{saPAKE}}$. Functionality $\mathcal{F}_{\text{saPAKE}}$ is a simple but significant variant of the UC aPAKE functionality $\mathcal{F}_{\text{aPAKE}}$ from [40], which is shown in Figure 4.2 and Figure 4.3 in Chapter 4. For an overview of the aPAKE functionality $\mathcal{F}_{\text{aPAKE}}$, see Section 4.2.

Strong aPAKE vs. aPAKE. Our functionality $\mathcal{F}_{\text{saPAKE}}$ is identical to $\mathcal{F}_{\text{aPAKE}}$ except that the underlined text in Figure 5.1 is omitted. (There is no change in the other part of the functionality, so we omit it.) That is, the only differences between $\mathcal{F}_{\text{saPAKE}}$ and $\mathcal{F}_{\text{aPAKE}}$ are in the actions upon the stealing of the password file; specifically, $\mathcal{F}_{\text{saPAKE}}$ omits recording the (offline, pw) pairs and does not allow for OfflineTestPwd queries made before the StealPwdFile query. Let us explain. Let's consider first the definition of $\mathcal{F}_{\text{saPAKE}}$, i.e., with the underlined text omitted. In this case, the actions upon server compromise, i.e., StealPwdFile, are simple. First, a flag (compromised) is defined to mark that the password file has been compromised. Second, once this event happens, the adversary is allowed to submit password guesses and be informed if a guess was correct. Note that each guess "costs" the attacker

Password Registration

- On $(\mathsf{StorePwdFile}, sid, \mathsf{C}, \mathsf{pw})$ from S, if this is the first StorePwdFile message, record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw} \rangle$ and mark it uncompromised.

Stealing Password Data

- On $(\mathsf{StealPwdFile}, sid)$ from $\mathcal{A}^*$, if there is no record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw} \rangle$, return "no password file" to $\mathcal{A}^*$. Otherwise, if the record is marked uncompromised, mark it compromised; regardless,
  - If there is a record $(\mathsf{offline}, \mathsf{pw})$, send pw to $\mathcal{A}^*$.
  - Else return "password file stolen" to $\mathcal{A}^*$.
- On $(\mathsf{OfflineTestPwd}, sid, \mathsf{pw}^*)$ from $\mathcal{A}^*$, do:
  - If there is a record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw} \rangle$ marked compromised, do: if $\mathsf{pw}^* = \mathsf{pw}$, return "correct guess" to $\mathcal{A}^*$; else return "wrong guess."
  - Else record $(\mathsf{offline}, \mathsf{pw})$.

Figure 5.1: Functionalities $\mathcal{F}_{\text{aPAKE}}$ (full text) and $\mathcal{F}_{\text{saPAKE}}$ (underlined text omitted) (the login phase is identical and thus omitted; see Figure 4.3)

one OfflineTestPwd query. This together with the restriction that these queries can only be made after the password file is compromised ensure that shortcuts in finding the password after such compromise are not possible, namely that the attacker needs to pay with one OfflineTestPwd query for each password it wants to test. Thus, pre-computation attacks are made infeasible.

Now consider the $\mathcal{F}_{\text{aPAKE}}$ functionality from [40] which includes the underlined text too. This functionality allows the attacker, via $(\mathsf{offline}, \mathsf{pw})$ records, to make guess queries against the password *even before the password file is compromised*. The restriction is that the responses to whether a guess was correct or not are provided to the attacker only after a StealPwdFile event. But note that if one of these guesses was correct, the attacker learns it *immediately* upon server compromise. This provision was necessary in [40] because the $\mathsf{file}[sid]$ in their aPAKE construction contains a deterministic publicly-computable hash of the password,

thus allowing for a pre-computation attack which lets the adversary instantaneously identify the password with a single table lookup upon server compromise.[2] Indeed, one can think of the pairs (offline, pw) in the original $\mathcal{F}_{\text{aPAKE}}$ functionality as a pre-computed table that the attacker builds overtime and which it can use to identify the password as soon as the server is compromised. By eliminating the ability to get guesses (offline, pw) answered before server compromise in our $\mathcal{F}_{\text{saPAKE}}$ functionality, we make such pre-computation attacks infeasible in the case of a strong aPAKE.

Modeling server compromise and offline dictionary queries. As in [40], we specify that StealPwdFile and OfflineTestPwd messages from $\mathcal{A}^*$ to $\mathcal{F}_{\text{saPAKE}}$ are accounted for by the environment. This is consistent with the UC treatment of adaptive compromise queries and is crucial to our modeling. Note that if the environment does not observe adaptive compromise queries then the ideal adversary, i.e., the simulator, could immediately corrupt all parties at the beginning of the protocol, learning their private inputs and thus making the work of simulation easier. By making the player-compromise queries (modeled by StealPwdFile command in our context) observable by the environment, we ensure that the environment's view of both the ideal and the real execution includes the same player-compromise events. This way we keep the simulator "honest," because it can only compromise a party if the environment accounts for it.

The same concern pertains to offline dictionary queries OfflineTestPwd, because if they were not observable by the environment, the ideal adversary could make such queries even if the real adversary does not. In particular, without environmental accounting for these queries the $\mathcal{F}_{\text{aPAKE}}$ and $\mathcal{F}_{\text{saPAKE}}$ functionalities would be equivalent because the simulator could internally gather all the offline dictionary attack queries made by the real-world adversary before server compromise, and it would send them all via the OfflineTestPwd query to $\mathcal{F}_{\text{saPAKE}}$

[2]Specifically, [40] stores a hash of the client's password in the password file, hence allowing an attacker to pre-compute a table with the hash of all passwords in a given dictionary and perform a simple table lookup upon compromising the password file.

after server compromise via the StealPwdFile query. Such simulator would make the ideal-world view indistinguishable from the real-world view to the environment *if* the environment does not observe the sequence of OfflineTestPwd and StealPwdFile queries.

5.3 Oblivious Pseudorandom Function

In the description below, we assume that $\mathsf{P} \in \{\mathsf{C}, \mathcal{A}^*\}$.
Public Parameter: PRF output-length ℓ, polynomial in security parameter κ.
For every server S, initialize $\mathsf{tickets}(\mathsf{S})$ to 0.
Initialization

- On (Init, sid, x) from server S, if this is the first Init message, mark S initialized, pick $F_{\mathsf{S}}(x) \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$ and send $(\mathsf{Init}, sid, F_{\mathsf{S}}(x))$ to S.

Server Compromise

- On $(\mathsf{Compromise}, sid)$ from $\mathcal{A}^*$ for a server S, if S is marked initialized, mark it compromised.
- On $(\mathsf{OfflineEval}, sid, \mathsf{S}, x)$ from $\mathcal{A}^*$, if S is corrupted or marked compromised, send $(\mathsf{OfflineEval}, sid, F_{\mathsf{S}}(x))$ to $\mathcal{A}^*$.

Evaluation

- On $(\mathsf{Eval}, sid, ssid, \mathsf{S}, x)$ from party P, if S is marked initialized, record $\langle ssid, \mathsf{P}, \mathsf{S}, x\rangle$ and send $(\mathsf{Eval}, sid, ssid, \mathsf{P}, \mathsf{S})$ to $\mathcal{A}^*$.
- On $(\mathsf{SvrComplete}, sid, ssid)$ from S, if S is marked initialized, send $(\mathsf{SvrComplete}, sid, ssid, \mathsf{S})$ to $\mathcal{A}^*$. Furthermore, if this is the first SvrComplete message for $ssid$, set $\mathsf{tickets}(\mathsf{S})$++.
- On $(\mathsf{RcvComplete}, sid, ssid, \mathsf{P}, \mathsf{S}^*)$ from $\mathcal{A}^*$, retrieve $\langle ssid, \mathsf{P}, \mathsf{S}, x\rangle$; abort if there is no such record, or $\mathsf{tickets}(\mathsf{S}^*) = 0$. Else set $\mathsf{tickets}(\mathsf{S}^*)$−− and send $(\mathsf{Eval}, sid, ssid, F_{\mathsf{S}^*}(x))$ to P (pick $F_{\mathsf{S}^*}(x) \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$ if undefined).

Figure 5.2: Revised OPRF functionality $\mathcal{F}_{\mathrm{OPRF}}$ with adaptive compromise

Oblivious Pseudorandom Functions (OPRF) are a central tool in all our constructions. For an overview of the OPRF and our 2HashDH protocol, see Chapter 3. Here we adopt the formulation from Section 3.2 as the basis for our revised OPRF functionality $\mathcal{F}_{\mathrm{OPRF}}$ presented

in Figure 5.2.

Changes from OPRF functionality of Section 3.2. To use UC OPRF in our applications we need to make some changes to the way functionality $\mathcal{F}_{\text{OPRF}}$ was defined in Section 3.2, as described below. Changes (2) and (3) are essentially syntactic and require only cosmetic changes in the security argument. Change (1) is the only one which influences the security argument in a more essential way. Fortunately, the 2HashDH protocol that we use for OPRF instantiation in our protocols, shown in Section 3.2 to realize the previous version of the OPRF functionality $\mathcal{F}_{\text{OPRF}}$, also realizes our modified functionality. We recall the 2HashDH protocol in Figure 5.3, adapting its syntax to our changes in $\mathcal{F}_{\text{OPRF}}$, and we argue that the security proof of Section 3.2 which shows that it realizes $\mathcal{F}_{\text{OPRF}}$ defined by Section 3.2 extends to the modified functionality $\mathcal{F}_{\text{OPRF}}$ presented here.

Let $H(\cdot,\cdot)$ and $H'(\cdot)$ be hash functions with ranges $\{0,1\}^\ell$ and $\mathbb{G}$, respectively (modeled as random oracles).
Initialization

- On input (Init, sid, x), if this is the first Init message, S picks and records $k \leftarrow_{\text{R}} \mathbb{Z}_m$, and outputs $(\mathsf{Init}, sid, H(x, H'(x)^k))$.

Evaluation

- On input $(\mathsf{Eval}, sid, ssid, \mathsf{S}, x)$, C proceeds as follows:
  - If there is a record $\langle \mathsf{S}, x, r, y\rangle$, C outputs $(\mathsf{Eval}, sid, ssid, y)$ to $\mathcal{Z}$.
  - Else if there is a record $\langle \mathsf{S}', x, r, y\rangle$ (where $\mathsf{S}' \neq \mathsf{S}$), C records $\langle \mathsf{S}, x, r, \perp\rangle$ and sends $a := H'(x)^r$ to S.
  - Else C picks $r \leftarrow_{\text{R}} \mathbb{Z}_m$, records $\langle \mathsf{S}, x, r, \perp\rangle$ and sends $a := H'(x)^r$ to S.
- On input $(\mathsf{SvrComplete}, sid, ssid)$ and a from C, S retrieves k and sends $b := a^k$ to C.
- On b from S, if this is the first such message for $ssid$, C retrieves record $\langle \mathsf{S}, x, r, \perp\rangle$, replaces $\perp$ with $y := H(x, b^{1/r})$ and outputs $(\mathsf{Eval}, sid, ssid, y)$.

Figure 5.3: Revised protocol 2HashDH (for PRF output length ℓ)

(1) We extend the OPRF functionality to allow the *adaptive compromise* of a server holding the PRF key via a Compromise message. Such action is needed in the aPAKE setting where the attacker $\mathcal{A}^*$ can compromise a server's password file that contains the server's OPRF key. After the compromise, $\mathcal{A}^*$ is allowed to compute that server's PRF function by itself on any value of its choice using an OfflineEval command and without the restrictions of the ticketing mechanism.

We note that functionality $\mathcal{F}_{\text{OPRF}}$ distinguishes between (statically) *corrupted servers* and (adaptively) *compromised sessions* (the latter representing different OPRF keys at the same server). This distinction allows for a granular separation between compromised and uncompromised OPRF keys held by the same server. We adopt this distinction for consistency with the aPAKE functionality from Figure 4.2 and Figure 4.3 that distinguishes between an entirely corrupted server and particular aPAKE instances that can be adaptively compromised by an adversary.

(2) We change the session-ID syntax used in Section 3.2 to model the use of multiple OPRF keys by the same server. In the formulation of Section 3.2 each PRF key was identified with a server identity making a one-to-one correspondence between OPRF keys and servers. Here, we allow multiple OPRF keys to be associated with one server. Each such key is identified with a tag *sid* and a server can be associated with multiple such tags. In the context of our application to aPAKE protocols, each aPAKE session is associated with a unique OPRF key used by the server for a particular client, so the session-ID *sid* corresponds to a client account at that server. Any *sid* can include *sub-sessions*, denoted by *ssid*, corresponding to different runs of the OPRF protocol between a client and a server.

(3) We add an initialization phase to the functionality, which models a server picking an OPRF key and, in addition, computing the OPRF value on any input. This interface simplifies the usage of OPRF in our applications to aPAKE, where the server will pick an

OPRF key for a new client and evaluate the OPRF on the client's password (for generating an encryption key). This modeling differs from Section 3.2 which frames OPRF initialization as an *interactive* procedure through an Eval call, while here it is performed *locally* by the server.

Theorem 7. *Suppose that the* $(Q + q_{\mathrm{H}'}, Q)$*-OMDH assumption holds for* $(\mathbb{G}, g, m)$*, where* Q *is the number of* Eval *messages sent to* C *and* $q_{\mathrm{H}'}$ *is the number of* $H'(\cdot)$ *queries. Then the revised protocol 2HashDH in Figure 5.3 realizes the revised* $\mathcal{F}_{\mathrm{OPRF}}$ *functionality in ROM.*

Proof. We only provide a proof sketch which briefly discusses how our modifications to $\mathcal{F}_{\mathrm{OPRF}}$ affect the security proof. Since no message is sent to $\mathcal{A}^*$ in the initialization phase, adding initialization does not affect the simulation. Allowing for sub-sessions (identified by *ssid*) results in adding *ssid* in the simulator whenever appropriate.

The remaining change is that $\mathcal{A}$ may compromise a server (for a specific *sid*) at any time; after that, $\mathcal{A}$ can compute the server's function value on any valid input. SIM is able to simulate this by sending OfflineEval messages to $\mathcal{F}$. Furthermore, note that fail may only occur on servers which are not marked compromised at that time; therefore, the argument upper-bounding Pr[fail] (in the setting where a server cannot be compromised) is unchanged. □

5.4 A Compiler from aPAKE to saPAKE via OPRF

In Figure 5.4 we specify a compiler that transforms any OPRF and any aPAKE into a strong aPAKE protocol. In UC terms the saPAKE protocol is defined in the $(\mathcal{F}_{\mathrm{OPRF}}, \mathcal{F}_{\mathrm{aPAKE}})$-hybrid world, for $\mathcal{F}_{\mathrm{OPRF}}$ with the output length parameter $\ell = 2\kappa$. The compiler is simple. First, the client transforms its password pw into a randomized value rw by interacting with the server in an OPRF protocol where the client inputs pw and the server inputs the OPRF key. Nothing is learned at the server about pw (i.e., rw is indistinguishable from random as

long as the input pw is not queried as input to the OPRF). Next, the client sets rw as its password in the given aPAKE protocol. Note that since the password rw is taken from a pseudorandom set, then even if the size of this set is the same as the original dictionary D from which pw was taken, the pseudorandom set is unknown to the attacker (the attacker can only learn this set via OPRF queries which require an online dictionary attack). Thus, any previous ability to run a pre-computation attack against the aPAKE protocol based on dictionary D is now lost.

We assume that $\mathcal{A}$ always simultaneously sends queries $(\mathsf{Compromise}, sid)$ and $(\mathsf{StealPwdFile}, sid)$ for the same sid to $\mathcal{F}_{\text{OPRF}}$ to $\mathcal{F}_{\text{aPAKE}}$, respectively, because in any instantiation of this protocol the server's OPRF-related state and aPAKE-related state would be part of the same $\mathsf{file}[sid]$. Consequently, for a single sid, S's status (compromised or not) in $\mathcal{F}_{\text{OPRF}}$ and $\mathcal{F}_{\text{aPAKE}}$ is always the same.

Password Registration

1. On input $(\mathsf{StorePwdFile}, sid, \mathsf{C}, \mathsf{pw})$, S sends $(\mathsf{Init}, sid, \mathsf{pw})$ to $\mathcal{F}_{\text{OPRF}}$.
   On $\mathcal{F}_{\text{OPRF}}$'s response $(\mathsf{Init}, sid, \mathsf{rw})$, S sends $(\mathsf{StorePwdFile}, sid, \mathsf{C}, \mathsf{rw})$ to $\mathcal{F}_{\text{aPAKE}}$.

Password Authentication and Key Generation

1. On input $(\mathsf{CltSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$, C sends $(\mathsf{Eval}, sid, ssid, \mathsf{S}, \mathsf{pw}')$ to $\mathcal{F}_{\text{OPRF}}$.
   On $\mathcal{F}_{\text{OPRF}}$'s response $(\mathsf{Eval}, sid, ssid, \mathsf{rw}')$, C sends $(\mathsf{CltSession}, sid, ssid, \mathsf{S}, \mathsf{rw}')$ to $\mathcal{F}_{\text{aPAKE}}$.
2. On input $(\mathsf{SvrSession}, sid, ssid)$, S sends $(\mathsf{SvrComplete}, sid, ssid)$ to $\mathcal{F}_{\text{OPRF}}$ and $(\mathsf{SvrSession}, sid, ssid)$ to $\mathcal{F}_{\text{aPAKE}}$.
3. On $(sid, ssid, SK)$ or $(\mathsf{abort}, sid, ssid)$ from $\mathcal{F}_{\text{aPAKE}}$, the recipient, either C or S, outputs this message.

Figure 5.4: Strong aPAKE protocol in the $(\mathcal{F}_{\text{OPRF}}, \mathcal{F}_{\text{aPAKE}})$-hybrid world

We claim the security of our protocol in the following theorem:

Theorem 8. *The protocol in Figure 5.4 realizes functionality $\mathcal{F}_{\text{saPAKE}}$ in the $(\mathcal{F}_{\text{OPRF}}, \mathcal{F}_{\text{aPAKE}})$-hybrid model.*

Proof. For any efficient environment $\mathcal{Z}$ and adversary $\mathcal{A}$ against the protocol, we construct a simulator SIM as in Figure 5.5 and Figure 5.6. Without loss of generality, suppose $\mathcal{A}$ is a "dummy" adversary who merely passes through all its messages to and from $\mathcal{Z}$, and all its computation to $\mathcal{Z}$. To keep notation brief we denote functionality $\mathcal{F}_{\text{saPAKE}}$ as $\mathcal{F}$.

For every server S, initialize $\mathsf{tickets}(\mathsf{S})$ to 0; initialize tested to $\varnothing$.
Stealing Password Data and Offline Queries

1. On $(\mathsf{Compromise}, sid)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{OPRF}}$ and $(\mathsf{StealPwdFile}, sid)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{aPAKE}}$, send $(\mathsf{StealPwdFile}, sid)$ to $\mathcal{F}$.
   If $\mathcal{F}$ returns "no password file," pass it to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.
   If $\mathcal{F}$ returns "password file stolen," mark S and $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \cdot \rangle$ $\mathsf{compromised}$ (record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \perp \rangle$ and mark it $\mathsf{compromised}$ if there is no such record). Furthermore, if $\cdot$ is a string rw and $\mathsf{rw} \in \mathsf{tested}$, then send rw to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$; else send "password file stolen" to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.
2. On $(\mathsf{OfflineEval}, sid, \mathsf{S}, x)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{OPRF}}$, if S is corrupted or marked $\mathsf{compromised}$, send $(\mathsf{OfflineEval}, sid, F_{\mathsf{S}}(x))$ to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{OPRF}}$ (pick $F_{\mathsf{S}}(x) \leftarrow_{\text{R}} \{0,1\}^{\ell}$ if it is undefined) and $(\mathsf{OfflineTestPwd}, sid, x)$ to $\mathcal{F}$. If $\mathcal{F}$ returns "correct guess," retrieve $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \cdot \rangle$ (there must be such record, since if S is marked $\mathsf{compromised}$, $\mathcal{A}$ must have sent $(\mathsf{Compromise}, sid)$ aimed at $\mathcal{F}_{\text{OPRF}}$ and $(\mathsf{StealPwdFile}, sid)$ aimed at $\mathcal{F}_{\text{aPAKE}}$ previously, and at that time $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \cdot \rangle$ was recorded); if the last item is $\perp$, replace it with $F_{\mathsf{S}}(x)$.
3. On message $(\mathsf{OfflineTestPwd}, sid, \mathsf{rw}^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{aPAKE}}$, add rw^* to tested. Furthermore, if there is a record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{rw} \rangle$ marked $\mathsf{compromised}$, do:
   - If $\mathsf{rw} = \mathsf{rw}^*$, send "correct guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.
   - Else send "wrong guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.

Figure 5.5: The simulator SIM for the aPAKE-based construction in the stealing password data phase

We now show that the distinguishing advantage of $\mathcal{Z}$ between the real world and the simulated world is negligible. As before, the argument uses a sequence of games, starting from the real world and ending at the simulated world.

$\mathbf{G}_0$ is the real world.

In $\mathbf{G}_1$, on $(\mathsf{CltSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$ from $\mathcal{Z}$ to C and $(\mathsf{RcvComplete}, sid, ssid, \mathsf{S}, \mathsf{S}^*)$ from

Password Authentication

1. On $(\mathsf{CltSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}$, send $(\mathsf{Eval}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{OPRF}}$. Also, if this is the first CltSession message for $ssid$, record $\langle ssid, \mathsf{C}, \mathsf{S}\rangle$ and mark it fresh.

2. On $(\mathsf{SvrSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}$, if there is no record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \cdot\rangle$, record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \perp\rangle$ and mark it uncompromised; regardless, send $(\mathsf{SvrComplete}, sid, ssid, \mathsf{S})$ and $(\mathsf{SvrSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$ as messages from resp. $\mathcal{F}_{\text{OPRF}}$ and $\mathcal{F}_{\text{aPAKE}}$. Also, if this is the first SvrSession message for $ssid$, set $\mathsf{tickets}(\mathsf{S})$++, record $\langle ssid, \mathsf{S}, \mathsf{C}\rangle$ and mark it fresh.

3. On $(\mathsf{RcvComplete}, sid, ssid, \mathsf{C}, \mathsf{S}^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{OPRF}}$, retrieve $\langle ssid, \mathsf{C}, \mathsf{S}\rangle$; ignore this message if such record does not exist, or $\mathsf{tickets}(\mathsf{S}^*) = 0$. Else set $\mathsf{tickets}(\mathsf{S}^*)--$, augment the record to $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{S}^*\rangle$, mark it fresh and send $(\mathsf{CltSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.

Active Session Attacks

1. On $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, \mathsf{rw}^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{aPAKE}}$, if there is a record $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{S}^*\rangle$ (if $\mathsf{P} = \mathsf{C}$) or $\langle ssid, \mathsf{S}, \mathsf{C}\rangle$ (if $\mathsf{P} = \mathsf{S}$) marked fresh, mark it unfresh and check if there is an x such that $\mathsf{rw}^* = F_{\mathsf{S}^*}(x)$ (if $\mathsf{P} = \mathsf{C}$) or $\mathsf{rw}^* = F_{\mathsf{S}}(x)$ (if $\mathsf{P} = \mathsf{S}$).

   - If there are more than one such x's, output collision and abort.
   - If there is a unique such x, send $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, x)$ to $\mathcal{F}$ and pass it to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.
     Also, if $\mathsf{P} = \mathsf{S}$ and $\mathcal{F}$ returns "correct guess", retrieve $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \cdot\rangle$, and if the last item is $\perp$, replace it with rw^*.
   - If there is no such x, send "wrong guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.

2. On $(\mathsf{Impersonate}, sid, ssid)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{aPAKE}}$, if there is a record $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{S}^*\rangle$ marked fresh, mark it unfresh and do:

   - If $\mathsf{S}^* = \mathsf{S}$, send $(\mathsf{Impersonate}, sid, ssid)$ to $\mathcal{F}$, and pass $\mathcal{F}$'s response ("correct guess" or "wrong guess") to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.
   - Else send "wrong guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.

Key Generation and Authentication

1. On $(\mathsf{NewKey}, sid, ssid, \mathsf{P}, SK^*)$ or $(\mathsf{TestAbort}, sid, ssid, \mathsf{P})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{aPAKE}}$, if there is a record $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{S}^*\rangle$ (if $\mathsf{P} = \mathsf{C}$) or $\langle ssid, \mathsf{S}, \mathsf{C}\rangle$ (if $\mathsf{P} = \mathsf{S}$) not marked completed, pass the message from $\mathcal{A}$ to $\mathcal{F}$. In the case of $(\mathsf{TestAbort}, sid, ssid, \mathsf{P})$, also pass $\mathcal{F}$'s response (succ or fail) to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$. Finally, mark the record above completed.

Figure 5.6: The simulator SIM for the aPAKE-based construction in the login phase

$\mathcal{A}$ to $\mathcal{F}_{\text{OPRF}}$, record $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{S}^*, \mathsf{rw}' \rangle$ (instead of $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{rw}' \rangle$). Obviously,

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_0, \mathbf{G}_1} = 0.$$

In $\mathbf{G}_2$, on $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, \mathsf{rw}^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{aPAKE}}$, if there is a record $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{S}^*, \mathsf{rw}' \rangle$ (if $\mathsf{P} = \mathsf{C}$) or $\langle ssid, \mathsf{S}, \mathsf{C}, \mathsf{rw} \rangle$ (if $\mathsf{P} = \mathsf{S}$) marked fresh, check if there is an x such that $\mathsf{rw}^* = F_{\mathsf{S}^*}(x)$ (if $\mathsf{P} = \mathsf{C}$) or $\mathsf{rw}^* = F_{\mathsf{S}}(x)$ (if $\mathsf{P} = \mathsf{S}$).

- If there are more than one such x's, output $\mathsf{collision}$ and abort.
- If there is a unique such x and $x = \mathsf{pw}'$ (if $\mathsf{P} = \mathsf{C}$) or $x = \mathsf{pw}$ (if $\mathsf{P} = \mathsf{S}$), send "correct guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.
- In all other cases (i.e., $x \neq \mathsf{pw}'/\mathsf{pw}$ or there is no such x), send "wrong guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{aPAKE}}$.

First consider event $\mathsf{collision}$. $\mathsf{collision}$ occurs if and only if there are more than one x's such that $\mathsf{rw}^* = F_{\mathsf{S}^*}(x)$ (if $\mathsf{P} = \mathsf{C}$) or $\mathsf{rw}^* = F_{\mathsf{S}}(x)$ (if $\mathsf{P} = \mathsf{S}$). This means that there are $x_1 \neq x_2$ such that $F_{\mathsf{S}^*}(x_1) = F_{\mathsf{S}^*}(x_2)$ or $F_{\mathsf{S}}(x_1) = F_{\mathsf{S}}(x_2)$. Note that $F_{\mathsf{S}}(\cdot)$ and $F_{\mathsf{S}^*}(\cdot)$ are both random functions onto $\{0,1\}^{2\kappa}$. Assuming that $\mathcal{A}$ sends q_{F} Eval and $\mathsf{OfflineEval}$ messages aimed at $\mathcal{F}_{\text{OPRF}}$ in total and there are q_{C} C sub-sessions, there are at most $q_{\text{F}} + q_{\text{C}} + 1$ F values defined in total (q_{F} defined by $\mathcal{A}$'s actions, q_{C} defined by C's input to the protocol, and 1 by S's input to the protocol), so we have that

$$\Pr[\mathsf{collision}] \leq \frac{(q_{\text{F}} + q_{\text{C}} + 1)^2}{2^{2\kappa+1}}.$$

Next assume that $\mathsf{collision}$ does not occur. Consider the first message of type $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, \mathsf{rw}^*)$ (note that $\mathcal{A}$ receives a reply for the first such message only, since $\langle ssid, \mathsf{P}, \mathsf{P}', \cdot \rangle$ becomes either $\mathsf{compromised}$ or $\mathsf{interrupted}$ after the first message). In

both $\mathbf{G}_1$ and $\mathbf{G}_2$, $\mathcal{A}$ receives "correct guess" if and only if $\mathsf{rw}^* = F_{\mathsf{S}^*}(\mathsf{pw}')$ (if $\mathsf{P} = \mathsf{C}$) or $\mathsf{rw}^* = F_{\mathsf{S}}(\mathsf{pw})$ (if $\mathsf{P} = \mathsf{S}$), so $\mathcal{Z}$'s views in $\mathbf{G}_1$ and $\mathbf{G}_2$ in this case are identical. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_1,\mathbf{G}_2} \leq \Pr[\mathsf{collision}] \leq \frac{(q_{\mathrm{F}} + q_{\mathrm{C}} + 1)^2}{2^{2\kappa+1}},$$

which is a negligible function of κ.

In $\mathbf{G}_3$, on $(\mathsf{Impersonate}, sid, ssid)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{aPAKE}}$, if there is a record $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{S}^*, \mathsf{rw}' \rangle$ marked fresh, send "correct guess" to $\mathcal{A}$ if S is marked $\mathsf{compromised}$, $\mathsf{S}^* = \mathsf{S}$ and $\mathsf{pw}' = \mathsf{pw}$; otherwise send "wrong guess."

Similar with above, $\mathcal{A}$ receives a reply for the first $\mathsf{Impersonate}$ message only, so we only consider the first such message. Note that in $\mathbf{G}_2$ $\mathcal{A}$ receives "correct guess" if and only if S is compromised and $\mathsf{rw}' = \mathsf{rw}$, where $\mathsf{rw}' = F_{\mathsf{S}^*}(\mathsf{pw}')$ and $\mathsf{rw} = F_{\mathsf{S}}(\mathsf{pw})$. $\mathcal{Z}$'s views in $\mathbf{G}_2$ and $\mathbf{G}_3$ are identical unless $(\mathsf{S}^* \neq \mathsf{S} \vee \mathsf{pw}' \neq \mathsf{pw}) \wedge F_{\mathsf{S}^*}(\mathsf{pw}') = F_{\mathsf{S}}(\mathsf{pw})$ (in which case $\mathcal{A}$ receives "correct guess" in $\mathbf{G}_2$ and "wrong guess" in $\mathbf{G}_3$). Since $\mathsf{S}^* \neq \mathsf{S}$ or $\mathsf{pw}' \neq \mathsf{pw}$, $F_{\mathsf{S}^*}(\mathsf{pw}')$ and $F_{\mathsf{S}}(\mathsf{pw})$ are two independent random strings is $\{0,1\}^{2\kappa}$; therefore, for a single C session, the probability that $F_{\mathsf{S}^*}(\mathsf{pw}') = F_{\mathsf{S}}(\mathsf{pw})$ is at most $1/2^{2\kappa}$. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_2,\mathbf{G}_3} \leq \frac{q_{\mathrm{C}}}{2^{2\kappa}},$$

which is a negligible function of κ.

In $\mathbf{G}_4$, after $\mathcal{Z}$ sends $(\mathsf{StorePwdFile}, sid, \mathsf{C}, \mathsf{pw})$ to S, record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \bot \rangle$ (instead of $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{rw} := F_{\mathsf{S}}(\mathsf{pw}) \rangle$); replace $\bot$ with $\mathsf{rw} := F_{\mathsf{S}}(\mathsf{pw})$ in the following two cases: (i) when $\mathcal{A}$ sends $(\mathsf{OfflineEval}, sid, \mathsf{S}, \mathsf{pw})$ to $\mathcal{F}_{\mathrm{OPRF}}$, and S is corrupted or marked $\mathsf{compromised}$; (ii) when $\mathcal{A}$ sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, \mathsf{rw}^*)$ to $\mathcal{F}_{\mathrm{aPAKE}}$, and $\mathsf{rw}^* = \mathsf{rw}$.

If neither (i) nor (ii) happens, $\mathsf{rw} = F_{\mathsf{S}}(\mathsf{pw})$ is a random string in $\{0,1\}^{2\kappa}$ in $\mathcal{Z}$'s view. Therefore, replacing rw with $\bot$ in this case creates a $1/2^{2\kappa}$ distinguishing advantage. We

have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_3,\mathbf{G}_4} \leq \frac{1}{2^{2\kappa}},$$

which is a negligible function of κ.

In $\mathbf{G}_5$, postpone the recording of $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \cdot \rangle$ until (i) $\mathcal{A}$ sends $(\mathsf{Compromise}, sid)$ to $\mathcal{F}_{\text{OPRF}}$ and $(\mathsf{StealPwdFile}, sid)$ to $\mathcal{F}_{\text{aPAKE}}$, or (ii) S sends $(\mathsf{SvrSession}, sid, ssid)$ to $\mathcal{F}_{\text{aPAKE}}$. Note that if neither (i) nor (ii) happens, $\mathbf{G}_4$ does not retrieve $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \cdot \rangle$. Therefore,

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_4,\mathbf{G}_5} = 0.$$

Note that in $\mathbf{G}_5$, $\mathsf{rw} = F_{\mathsf{S}}(\mathsf{pw})$ and $\mathsf{rw}' = F_{\mathsf{S}^*}(\mathsf{pw}')$ are defined no matter $\mathcal{A}$ queries them (i.e., $\mathcal{A}$ sends $(\mathsf{OfflineEval}, sid, \mathsf{S}, \mathsf{pw})$ to $\mathcal{F}_{\text{OPRF}}$ when S is corrupted or marked $\mathsf{compromised}$; or $\mathcal{A}$ sends $(\mathsf{Eval}, sid, ssid, \mathsf{pw}')$ and then $(\mathsf{RcvComplete}, sid, ssid, \mathcal{A}, \mathsf{S}^*)$ to $\mathcal{F}_{\text{OPRF}}$) or not. In $\mathbf{G}_6$, leave rw (resp. rw') undefined unless and until $\mathcal{A}$ queries $F_{\mathsf{S}}(\mathsf{pw})$ (resp. $F_{\mathsf{S}^*}(\mathsf{pw}')$).

If $\mathcal{A}$ does not query $F_{\mathsf{S}}(\mathsf{pw})$ (resp. $F_{\mathsf{S}^*}(\mathsf{pw}')$), rw (resp. rw') is a random string in $\{0,1\}^{2\kappa}$ in $\mathcal{Z}$'s view. Since there is 1 rw and q_{C} rw''s, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_5,\mathbf{G}_6} \leq \frac{q_{\text{C}} + 1}{2^{2\kappa}},$$

which is a negligible function of κ.

$\mathbf{G}_7$ is the simulated world. We can see that the change from $\mathbf{G}_6$ to $\mathbf{G}_7$ is merely conceptual, with the game challenger split into the saPAKE functionality $\mathcal{F}$ and the simulator SIM. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_6,\mathbf{G}_7} = 0.$$

Summing up all results above, we conclude that $\mathcal{Z}$'s distinguishing advantage between the real world and the simulated world is a negligible function of κ. This completes the proof. □

5.5 A Compiler from AKE-KCI to saPAKE via OPRF

Our second transformation for building a strong aPAKE protocol composes an OPRF with an Authenticated Key Exchange (AKE) protocol, "glued" together using an authenticated encryption scheme. We require the AKE to be secure in the UC framework, namely, to realize the UC KE functionality of [28], and to also be "KCI secure." The latter notion was defined in [56] under a game-based formulation and formalized in Section 5.5.1 below in the UC framework. We only prove the OPRF-AKE composition to be saPAKE secure for protocols where the last party to compute the key in the AKE protocol is the server. The case where the client is the last party to compute the key (e.g., using a 2-message AKE protocol without explicit client authentication) needs relaxation of the $\mathcal{F}_{\text{saPAKE}}$ functionality, and we intend to explore this case as future work.

5.5.1 UC Definition of AKE-KCI

The KCI notion for KE protocols, which stands for "Key-Compromise Impersonation," captures the property we call "security against reverse impersonation," which concerns an attacker $\mathcal{A}$ who learns party P's long-term keys but otherwise does not actively control P. Resistance to KCI attacks, or "KCI security" for short, postulates that even though $\mathcal{A}$ can impersonate P to other parties, sessions which P itself runs with honest peers need to remain secure. A game-based definition of this notion appears in [56], and here we formalize it in the UC framework through functionality $\mathcal{F}_{\text{AKE}-\text{KCI}}$ presented in Figure 5.7.

Functionality $\mathcal{F}_{\text{AKE}-\text{KCI}}$ extends the standard KE functionality of [28] with two adversarial actions. The first, Compromise, captures the compromise of a party's keys. The second is Impersonate which is borrowed from the aPAKE functionality of [40] shown in Figure 4.2 and Figure 4.3. This action is targeted for some party's sessions with its peer compromised

In the description below, we assume $\mathsf{P} \in \{\mathsf{C}, \mathsf{S}\}$.

- On $(\mathsf{CltSession}, sid, ssid, \mathsf{S})$ from C, send $(\mathsf{CltSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}^*$. If it is the first CltSession message for $ssid$, record $(ssid, \mathsf{C}, \mathsf{S})$ and mark it fresh.
- On $(\mathsf{SvrSession}, sid, ssid, \mathsf{C})$ from S, send $(\mathsf{SvrSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}^*$. If it is the first SvrSession message for $ssid$, record $(ssid, \mathsf{S}, \mathsf{C})$ and mark it fresh.
- On $(\mathsf{Compromise}, sid, \mathsf{P})$ from $\mathcal{A}^*$, mark P compromised.
- On $(\mathsf{Impersonate}, sid, ssid, \mathsf{P})$ from $\mathcal{A}^*$, if P is marked compromised and there is a record $(ssid, \mathsf{P}, \mathsf{P}')$ marked fresh, mark the record compromised.
- On $(\mathsf{NewKey}, sid, ssid, \mathsf{P}, SK^*)$ from $\mathcal{A}^*$ where $|SK^*| = \kappa$, if there is a record $(ssid, \mathsf{P}, \mathsf{P}')$ not marked completed, do:
  - If the record is marked compromised, or either P or P' is corrupted, set $SK := SK^*$.
  - If the record is marked fresh, a $(sid, ssid, SK')$ tuple was sent to P', and at that time there was a record $(ssid, \mathsf{P}', \mathsf{P})$ marked fresh, set $SK := SK'$.
  - Else pick $SK \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$.

  Finally, mark $(ssid, \mathsf{P}, \mathsf{P}')$ completed and send $(sid, ssid, SK)$ to P.

Figure 5.7: Functionality $\mathcal{F}_{\mathrm{AKE-KCI}}$

via the Compromise action, and it marks such session as compromised, which implies that the attacker can determine the session key this session outputs via the NewKey message. This models the fact that for a compromised party, its peer's sessions cannot be assumed to be secure since they could have been run with the adversary who has stolen the server's keys. However, sessions at the party itself must not be affected by the Impersonate action, and they remain secure. All other elements in $\mathcal{F}_{\mathrm{AKE-KCI}}$ are the same as in the basic UC KE functionality, except of some syntactic specialization to the client-server setting.

AKE-KCI security of HMQV. Our concrete instantiation of an saPAKE protocol, OPAQUE (Figure 5.11 in Section 5.6), is illustrated with 3-message HMQV [56] (with explicit client authentication) as the AKE-KCI component. The KCI property of HMQV was proved in [56] in the game-based Canetti-Krawczyk model [27] extended to include

KCI security. Here we require UC security, namely, a protocol that realizes functionality $\mathcal{F}_{\text{AKE-KCI}}$. Fortunately, [28] proves the equivalence of the game-based definition of [27] and their UC AKE formulation. Thanks to this equivalence, HMQV, as a basic KE, is secure in the UC framework. More precisely, this applies to the 3-message HMQV (which satisfies the "ACK" property required for the equivalence in [28]). For our purposes, however, we need HMQV to realize the extended AKE-KCI functionality of Figure 5.7. The equivalence with the game-based definition extends to this case. Indeed, since the original equivalence from [28] holds even in the case of adaptive party corruptions, the Compromise and Impersonate actions introduced here – which constitute a *limited* form of adaptive corruptions – follow as a special case. Finally, we note that the equivalence between the above models also preserves forward secrecy, so this property (proved in the game-based Canetti-Krawczyk model in [56]) holds in the UC too. We note that by the results in [56], the 3-message HMQV enjoys full perfect forward secrecy. The above security of HMQV (without including security against the leakage of ephemeral exponents) is based on the CDH assumption in ROM [56].

AKE-KCI with server as last-to-complete party. The UC AKE-KCI protocol (which realizes $\mathcal{F}_{\text{AKE-KCI}}$) above, composed with the UC OPRF protocol, does not necessarily yield a UC saPAKE. Indeed, suppose that the AKE-KCI allows S to output its key and end its session before C computes the OPRF output. Then when an adversarial client C^* tests a candidate password pw^* (i.e., upon receiving S's ciphertext c, C^* computes the OPRF on pw^* and uses the result to decrypt S's ciphertext), at the time when the simulator is able to extract pw^* from C^*'s OPRF computation, S's session is already completed. Hence the simulator fails to complete its simulation as it cannot send a TestPwd message to the saPAKE functionality $\mathcal{F}_{\text{saPAKE}}$ (recall that in $\mathcal{F}_{\text{saPAKE}}$, TestPwd queries on a completed session do not have any effect).

To overcome this technical difficulty, we require that in the AKE protocol, the client completes its session and gets its session key before the server does. Formally, we modify the UC functionality $\mathcal{F}_{\text{AKE-KCI}}$ to $\mathcal{F}_{\text{AKE-KCI+}}$ by adding the following line:

- On $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, SK^*)$ from $\mathcal{A}^*$, ignore this message if there is no $(sid, ssid, \mathsf{C})$ session marked completed.

It is clear that the 3-message HMQV realizes the $\mathcal{F}_{\text{AKE-KCI+}}$ functionality, since the server waits for the client's authentication message before it outputs its session key.

5.5.2 Strong aPAKE Construction from OPRF and AKE-KCI

Our saPAKE protocol based on OPRF and AKE-KCI is shown in Figure 5.8. The protocol uses the same OPRF tool as the saPAKE construction of Section 5.4, for length parameter $\ell = 2\kappa$, which defines the "randomized password" value $\mathsf{rw} = F_k(\mathsf{pw})$ for client C's password pw and OPRF key k held by server S. We assume that in the AKE-KCI protocol Π each party holds a (private, public) key pair, and that the each party runs the login subprotocol using its key pair and the public key of the counterparty as inputs. In the password registration phase, server S generates the client C's keys, and S's password file contains S's key pair (p_s, P_s); C's public key P_c; and a ciphertext c of C's private key p_c and the public keys P_c and P_s created using an authenticated encryption scheme with $\mathsf{rw} = F_k(\mathsf{pw})$ as the key. After creating the password file, value p_c is erased at S. In the login phase, S runs OPRF with C, which lets C compute $\mathsf{rw} = F_k(\mathsf{pw})$, it sends c to C, who can decrypt it under rw and retrieves its key pair (p_c, P_c) together with the server's key P_s, at which point both parties have appropriate inputs to the AKE-KCI protocol Π to compute the session key.

Public Components

- KCI-secure AKE protocol Π with private/public keys denoted p_s, P_s, p_c, P_c;
- Random-key robust authenticated encryption $\mathsf{AE} = (\mathsf{KeyGen}, \mathsf{AuthEnc}, \mathsf{AuthDec})$ where $\mathsf{KeyGen}(1^\kappa)$ picks a key uniformly at random from $\{0,1\}^{2\kappa}$;
- Functionality $\mathcal{F}_{\text{OPRF}}$ with output length parameter $\ell = 2\kappa$.

Password Registration

1. On input $(\mathsf{StorePwdFile}, sid, \mathsf{C}, \mathsf{pw})$, S generates pairs (p_s, P_s) and (p_c, P_c), and sends $(\mathsf{Init}, sid, \mathsf{pw})$ to $\mathcal{F}_{\text{OPRF}}$.
   On $\mathcal{F}_{\text{OPRF}}$'s response $(\mathsf{Init}, sid, \mathsf{rw})$, S computes $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_c, P_c, P_s)$ and records $\mathsf{file}[sid] := \langle p_s, P_s, P_c, c\rangle$.

Server Compromise

1. On $(\mathsf{StealPwdFile}, sid)$ from $\mathcal{A}$, S retrieves $\mathsf{file}[sid]$ and sends it to $\mathcal{A}$ (if there is no such record, sends "no password file" to $\mathcal{A}$).

Login

1. On $(\mathsf{CltSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$, C sends $(\mathsf{Eval}, sid, ssid, \mathsf{S}, \mathsf{pw}')$ to $\mathcal{F}_{\text{OPRF}}$.
2. On $(\mathsf{SvrSession}, sid, ssid)$, S retrieves $\mathsf{file}[sid] = \langle p_s, P_s, P_c, c\rangle$, sends c to C and $(\mathsf{SvrComplete}, sid, ssid)$ to $\mathcal{F}_{\text{OPRF}}$, and runs Π on input (p_s, P_s, P_c).
3. On $(\mathsf{Eval}, sid, ssid, \mathsf{rw}')$ from $\mathcal{F}_{\text{OPRF}}$ and c from S, C computes $\mathsf{AuthDec}_{\mathsf{rw}'}(c)$. If the result is $\perp$, C outputs $(\mathsf{abort}, sid, ssid)$ and halts. Else C parses $(p_c^*, P_c^*, P_s^*) := \mathsf{AuthDec}_{\mathsf{rw}'}(c)$ and runs Π on input (p_c^*, P_c^*, P_s^*).
4. Given Π's local output SK, the corresponding party, either C or S, outputs $(sid, ssid, SK)$.

Figure 5.8: Strong aPAKE based on AKE-KCI in the $\mathcal{F}_{\text{OPRF}}$-hybrid world

Role of authenticated encryption. The saPAKE protocol of Figure 5.8 utilizes an authenticated encryption scheme $\mathsf{AE} = (\mathsf{KeyGen}, \mathsf{AuthEnc}, \mathsf{AuthDec})$ to encrypt and authenticate C's AKE "credential" (p_c, P_c, P_s). We encrypt the whole payload (p_c, P_c, P_s) for simplicity, because unlike C's private key p_c, values P_c and P_s could be public and need to be only authenticated, not encrypted. However, the authentication property of AE must apply to the whole payload. Intuitively, C must authenticate S's public key P_s, but if C derived even its key pair (p_c, P_c) using just the secrecy of $\mathsf{rw} = F_k(\mathsf{pw})$, e.g., using rw as

randomness in a key generation, and C then executed AKE on such (p_c, P_c) pair, the resulting protocol would already be insecure. To see an example, if an AKE leaks C's public key input P_c (note that AKE does not guarantee privacy of the public key) then an adversary who engages C in a single protocol instance can find C's password pw via an offline dictionary attack by running the OPRF with C on some key k^*, and then given P_c leaked in the subsequent AKE it finds pw such that the key generation outputs P_c as a public key on randomness $\mathsf{rw} = F_{k^*}(\mathsf{pw})$.

Thus the role of the authentication property in AE is to commit $\mathcal{A}$ to a single guess of rw and consequently, given the OPRF key k^*, to a single guess pw. (Note that our UC OPRF notion implies that F is collision-resistant.) To that end we need AE to satisfy the following property which we call *random-key robustness*:[3] For any efficient algorithm $\mathcal{A}$ the probability that $\mathcal{A}$ on input (k_1, k_2) for two random keys k_1 and k_2 outputs c such that $\mathsf{AuthDec}_{k_1}(c) \neq \perp$ and $\mathsf{AuthDec}_{k_2}(c) \neq \perp$ is negligible. In other words, it must be infeasible to create an authenticated ciphertext that successfully decrypts under two different randomly generated keys. This property can be achieved in the standard model using e.g., encrypt-then-MAC with a MAC that is collision resistant with respect to the message and key, a property enjoyed by HMAC with full hash output. In ROM used by our saPAKE application one can also enforce it for any authenticated encryption scheme by attaching to its ciphertext c a hash $H(k, c)$ for a RO hash $H(\cdot, \cdot)$ with 2κ-bit outputs.

Note on not utilizing $\mathcal{F}_{\text{AKE-KCI+}}$. In Figure 5.8 we abstract the OPRF protocol as functionality $\mathcal{F}_{\text{OPRF}}$, but we use the real-world AKE-KCI protocol Π, rather than functionality $\mathcal{F}_{\text{AKE-KCI+}}$. The reason for this presentation is that in the KE functionality of [28], of which $\mathcal{F}_{\text{AKE-KCI}}$ is an extension, it is not clear how to support a usage of the KE protocol on keys which are computed via some other mechanism than the intended KE key

[3] This notion is a weakening of *Full Robustness (FROB)* from [35] where the attacker is allowed to choose k_1 and k_2 (in our case these keys are random). An even weaker notion, Semi-FROB, is defined in [35] where k_1 and k_2 are random but only k_1 is provided to $\mathcal{A}$.

generation. The KE functionality of [28] assumes that each entity keeps its private key as a permanent state, authenticates to a counterparty given its identity, and a KE party cannot specify any bitstring as one's own private key and a counterparty's public key. This is not how we use AKE in our saPAKE of Figure 5.8 precisely because the client C does not keep state and has to reconstruct its keys from a password (via OPRF). However, we can still use the real-world protocol Π, which UC realizes $\mathcal{F}_{\text{AKE-KCI+}}$, giving it the OPRF-computed information as input. In the proof of security we utilize the simulator $\mathsf{SIM}_{\text{AKE}}$, which shows that Π UC realizes $\mathcal{F}_{\text{AKE-KCI}}$, in our simulator construction, but we rely on its correctness only if C runs Π on the correctly reconstructed (p_c, P_s, P_s), and if the adversary causes C to reconstruct a different string we interpret this as a successful attack on C's login session.

5.5.3 Proof of Security

Theorem 9. *If protocol* Π *realizes functionality* $\mathcal{F}_{\text{AKE-KCI+}}$*, then the protocol in Figure 5.8 realizes functionality* $\mathcal{F}_{\text{saPAKE}}$ *in the* $\mathcal{F}_{\text{OPRF}}$*-hybrid model.*

The proof of this theorem is more complex than the previous proofs, so we provide an overview of the simulation strategy first. First consider password file storage (i.e., offline security). The simulator SIM generates the two key pairs (p_c, P_c) and (p_s, P_s), and instead of computing $\mathsf{rw} := F_{\mathsf{S}}(\mathsf{pw})$, it picks rw at random. Then it computes $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_c, P_c, P_s)$ and records $\mathsf{file}[sid] := \langle p_s, P_s, P_c, c\rangle$, just as what the real S does. The only way for the environment $\mathcal{Z}$ to learn information about rw is to let the adversary $\mathcal{A}$ query $F_{\mathsf{S}}(\mathsf{pw})$, either offline (i.e., $\mathcal{A}$ compromises S and sends $(\mathsf{OfflineEval}, sid, \mathsf{S}, \mathsf{pw})$ aimed at $\mathcal{F}_{\text{OPRF}}$) or online (i.e., $\mathcal{A}$ sends $(\mathsf{Eval}, sid, ssid, \mathsf{S}, x)$ and then $(\mathsf{RcvComplete}, sid, ssid, \mathcal{A}, \mathsf{S})$ aimed at $\mathcal{F}_{\text{OPRF}}$); in both cases, SIM is able to "program" the result as rw.

In the login phase (i.e., online security), the key observation is that C outputs $(\mathsf{abort}, sid, ssid)$ with overwhelming probability, except if either of the following happens

(let c^* be the ciphertext from $\mathcal{A}$ to C):

- (i) $\mathcal{A}$ is passive until the execution of Π begins, i.e., $\mathsf{S}^* = \mathsf{S}$ and $c^* = c$, and (ii) the two parties' passwords match, i.e., $\mathsf{pw}' = \mathsf{pw}$ (SIM can test if $\mathsf{pw}' = \mathsf{pw}$ via a $\mathsf{TestAbort}$ message). In this case the two parties' inputs in Π match, i.e., C's input is (p_c, P_c, P_s) and S's input is (p_s, P_s, P_c), so SIM "outsources" the simulation of Π to $\mathsf{SIM}_{\mathrm{AKE}}$. Also note that regarding the statuses of the two parties in the execution of Π,
  - If $\mathcal{A}$ queries $\mathsf{rw} = F_{\mathsf{S}}(\mathsf{pw})$, then $\mathcal{Z}$ learns p_c (as part of $\mathsf{AuthDec}_{\mathsf{rw}}(c)$). This is equivalent to C is compromised in Π. On the other hand, if $\mathcal{A}$ does not query $F_{\mathsf{S}}(\mathsf{pw})$, then $\mathsf{rw} = \mathsf{rw}' = F_{\mathsf{S}}(\mathsf{pw})$ is a random string in the environment $\mathcal{Z}$'s view, so by security of AE, $\mathcal{Z}$ learns no information about p_c.
  - If $\mathcal{A}$ compromises S, then $\mathcal{Z}$ learns p_s (as part of $\mathsf{file}[sid]$). This is equivalent to S is compromised in Π. On the other hand, if $\mathcal{A}$ does not compromise S, then $\mathcal{Z}$ learns no information about p_s.

  Since $\mathcal{A}$'s abilities of compromising S and performing offline attacks are adaptive, we need to require that in the security notion of AKE, both parties can be compromised adaptively. In the proof below, we mark this case $(*)$.
- $\mathcal{A}$ queries $\mathsf{rw}' = F_{\mathsf{S}^*}(\mathsf{pw}')$ (e.g., by running another sub-session of OPRF with S^* as the server, or in the case that $\mathsf{S}^* = \mathsf{S}$ and $\mathsf{pw}' = \mathsf{pw}$, by compromising S and computing $F_{\mathsf{S}}(\mathsf{pw})$ offline). In this case $\mathcal{Z}$ can choose its own (p_c^*, P_c^*, P_s^*) and encrypt $c^* \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}'}(p_c^*, P_c^*, P_s^*)$. However, SIM, who sees $\mathcal{A}$'s queries, can learn the same information as well, so it can run Π on C's behalf, hence simulate C's behavior in Π. (Here the random-key robustness of AE is needed, since if c^* is valid under two different keys, SIM does not know the plaintext under which key is (p_c^*, P_c^*, P_s^*).) Similarly, SIM can also run Π on S's behalf. In the proof below, we mark this case $(**)$.

We now proceed to the formal proof.

For every server S, initialize $\mathsf{tickets}(\mathsf{S})$ to 0.
Generate two key pairs (p_s, P_s) and (p_c, P_c), pick $\mathsf{rw} \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$, compute $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_c, P_c, P_s)$, and record $\mathsf{file}[sid] := \langle p_s, P_s, P_c, c\rangle$.
Let Π_c and Π_s be C and S's algorithm in the execution of Π, respectively.
Stealing Password Data and Offline Queries

1. On $(\mathsf{Compromise}, sid)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{OPRF}}$ and $(\mathsf{StealPwdFile}, sid)$ from $\mathcal{A}$ aimed at S, send $(\mathsf{StealPwdFile}, sid)$ to $\mathcal{F}$.
   If $\mathcal{F}$ returns "no password file," pass this message to $\mathcal{A}$ as a message from S.
   If $\mathcal{F}$ returns "password file stolen," mark S $\mathsf{compromised}$ and send $\mathsf{file}[sid]$ to $\mathcal{A}$ as a message from S.
2. On $(\mathsf{OfflineEval}, sid, \mathsf{S}, x)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{OPRF}}$, if S is marked $\mathsf{compromised}$ or corrupted, send $(\mathsf{OfflineTestPwd}, sid, x)$ to $\mathcal{F}$. If $\mathcal{F}$ returns "correct guess," record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, x\rangle$ and set $F_{\mathsf{S}}(x) := \mathsf{rw}$. Regardless, send $(\mathsf{OfflineEval}, sid, F_{\mathsf{S}}(x))$ to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathrm{OPRF}}$ (pick $F_{\mathsf{S}}(x) \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$ if undefined).

Figure 5.9: The simulator SIM for the AKE-based construction in the stealing password data phase

Proof. For any efficient environment $\mathcal{Z}$ and efficient adversary $\mathcal{A}$ against the protocol, we construct a simulator SIM as in Figure 5.9 and Figure 5.10. Since Π realizes $\mathcal{F}_{\mathrm{AKE-KCI+}}$, there is a simulator $\mathsf{SIM}_{\mathrm{AKE}}$ which produces a view indistinguishable with the view in the real execution of Π for any efficient environment and adversary; SIM invokes $\mathsf{SIM}_{\mathrm{AKE}}$ as a black box (while interacting with $\mathsf{SIM}_{\mathrm{AKE}}$, SIM plays the role of both $\mathcal{F}_{\mathrm{AKE-KCI}}$ and $\mathcal{A}$). Without loss of generality, suppose $\mathcal{A}$ is a "dummy" adversary who merely passes through all its messages to and from $\mathcal{Z}$, and all its computation to $\mathcal{Z}$. To keep notation brief we denote functionality $\mathcal{F}_{\mathrm{saPAKE}}$ as $\mathcal{F}$.

We now show that the distinguishing advantage of $\mathcal{Z}$ between the real world and the simulated world is negligible. As before, the argument uses a sequence of games, starting from the real world and ending at the simulated world.

$\mathbf{G}_0$ is the real world.

In $\mathbf{G}_1$, at the beginning of the game, pick $\mathsf{rw} \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$ (instead of $\mathsf{rw} := F_{\mathsf{S}}(\mathsf{pw})$) and

Password Authentication

1. On $(\mathsf{CltSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}$, send $(\mathsf{Eval}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$ as a message from $\mathcal{F}_{\text{OPRF}}$. Also, if this is the first CltSession message for $ssid$, record $\langle ssid, \mathsf{C}, \mathsf{S}\rangle$.

2. On $(\mathsf{SvrSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}$, retrieve $\mathsf{file}[sid] = \langle p_s, P_s, P_c, c\rangle$, send c and $(\mathsf{SvrComplete}, sid, ssid, \mathsf{S})$ to $\mathcal{A}$ as a message from S to C and from $\mathcal{F}_{\text{OPRF}}$, respectively, and send $(\mathsf{SvrSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM}_{\text{AKE}}$. Also, if this is the first SvrSession message for $ssid$, set $\mathsf{tickets}(\mathsf{S})$++.

3. On $(\mathsf{RcvComplete}, sid, ssid, \mathsf{C}, \mathsf{S}^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{OPRF}}$, retrieve $\langle ssid, \mathsf{C}, \mathsf{S}\rangle$; ignore this message if such record does not exist, or $\mathsf{tickets}(\mathsf{S}^*) = 0$. Else set $\mathsf{tickets}(\mathsf{S}^*)$−−, augment $\langle ssid, \mathsf{C}, \mathsf{S}\rangle$ to $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{S}^*\rangle$ and mark $(ssid, \mathsf{C})$ completed.

Key Generation and Authentication

1. On $(\mathsf{Eval}, sid, ssid, \mathsf{S}, x)$ and $(\mathsf{RcvComplete}, sid, ssid, \mathcal{A}, \mathsf{S})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\text{OPRF}}$, in addition to simulating $\mathcal{F}_{\text{OPRF}}$ by running its code, send $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, x)$ to $\mathcal{F}$. If $\mathcal{F}$ returns "correct guess," record $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, x\rangle$ and set $F_{\mathsf{S}}(x) := \mathsf{rw}$.

2. As soon as $(ssid, \mathsf{C})$ is marked completed and a c^* is sent from $\mathcal{A}$ aimed at C, retrieve $\mathsf{file}[sid] = \langle p_s, P_s, P_c, c\rangle$ and $\langle ssid, \mathsf{C}, \mathsf{S}, \mathsf{S}^*\rangle$ and proceed as follows:

   - If $c^* = c$ and $\mathsf{S}^* = \mathsf{S}$, send $(\mathsf{TestAbort}, sid, ssid, \mathsf{C})$ to $\mathcal{F}$.
     If $\mathcal{F}$ returns succ, send $(\mathsf{CltSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM}_{\text{AKE}}$ and do: (1) when S is marked compromised, send $(\mathsf{Compromise}, sid, \mathsf{S})$ to $\mathsf{SIM}_{\text{AKE}}$; when a $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw}\rangle$ is recorded, send $(\mathsf{Compromise}, sid, \mathsf{C})$ to $\mathsf{SIM}_{\text{AKE}}$; (2) on $(\mathsf{Impersonate}, sid, ssid, \mathsf{S})$ from $\mathsf{SIM}_{\text{AKE}}$, pass this message to $\mathcal{F}$; on $(\mathsf{Impersonate}, sid, ssid, \mathsf{C})$ from $\mathsf{SIM}_{\text{AKE}}$, send $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, \mathsf{pw})$ to $\mathcal{F}$; (3) while $\mathsf{SIM}_{\text{AKE}}$ simulates the execution of Π, pass messages between it and $\mathcal{A}$; (4) on $(\mathsf{NewKey}, sid, ssid, \mathsf{P}, SK)$ from $\mathsf{SIM}_{\text{AKE}}$, pass this message to $\mathcal{F}$. Mark this case $(*)$.
   - Else for every x such that $F_{\mathsf{S}^*}(x)$ is defined (denote it y), check if $\mathsf{AuthDec}_y(c^*) \neq \perp$.
     - If there are more than one such x's, output halt and abort.
     - If there is a unique such x, send $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, x)$ to $\mathcal{F}$.
       If $\mathcal{F}$ returns "correct guess," parse $(p_c^*, P_c^*, P_s^*) := \mathsf{AuthDec}_y(c^*)$ and do: (1) on $\mathcal{A}$'s message as from S to C in the execution of Π, run Π_c on (p_c^*, P_c^*, P_s^*); (2) when Π_c is completed with output SK, send $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, SK)$ to $\mathcal{F}$. Mark this case $(**)$.
       If $\mathcal{F}$ returns "wrong guess," send $(\mathsf{TestAbort}, sid, ssid, \mathsf{C})$ to $\mathcal{F}$.
     - If there is no such x, send $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, \perp)$ and then $(\mathsf{TestAbort}, sid, ssid, \mathsf{C})$ to $\mathcal{F}$.

3. If case $(*)$ does not happen, no matter whether case $(**)$ happens or not: (1) when a $\langle \mathsf{file}, \mathsf{C}, \mathsf{S}, \mathsf{pw}\rangle$ is recorded, send $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, \mathsf{pw})$ to $\mathcal{F}$; (2) on $\mathcal{A}$'s message as from C to S in the execution of Π, run Π_s on (p_s, P_s, P_c); (3) when Π_s is completed with output SK, send $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, SK)$ to $\mathcal{F}$.

Figure 5.10: The simulator SIM for the AKE-based construction in the login phase

leave $F_{\mathsf{S}}(\mathsf{pw})$ undefined. In addition, if (1) $\mathcal{A}$ sends $(\mathsf{Compromise}, sid)$ and then $(\mathsf{OfflineEval}, sid, \mathsf{S}, \mathsf{pw})$ to $\mathcal{F}_{\mathrm{OPRF}}$, or (2) $\mathcal{A}$ sends $(\mathsf{Eval}, sid, ssid, \mathsf{S}, \mathsf{pw})$ and then $(\mathsf{RcvComplete}, sid, ssid, \mathcal{A}, \mathsf{S})$ to $\mathcal{F}_{\mathrm{OPRF}}$, set $F_{\mathsf{S}}(\mathsf{pw}) := \mathsf{rw}$.

Observe that rw is a random string in $\{0,1\}^{2\kappa}$ in $\mathcal{Z}$'s view unless and until either (1) or (2) occurs; therefore, this modification does not change the distribution of rw in $\mathcal{Z}$'s view, so we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_0,\mathbf{G}_1} = 0.$$

$\mathbf{G}_2$ considers the case that (i) $c^* = c$ and $\mathsf{S}^* = \mathsf{S}$, (ii) $\mathsf{pw}' = \mathsf{pw}$, and (iii) *throughout the execution of* Π, $F_{\mathsf{S}}(\mathsf{pw})$ is undefined. (As in the description of SIM, we mark the combination of (i) and (ii) as $(*)$.) In this case replace c with a "dummy" value $\tilde{c} \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(\tilde{p}_c, \tilde{P}_c, P_s)$ where $(\tilde{p}_c, \tilde{P}_c)$ is a freshly generated key pair independently random of everything else.

Note that rw is a random string in $\{0,1\}^{2\kappa}$. By a reduction $\mathcal{R}_{\mathrm{SEC}}$ to the security of AE, for a single C sub-session, the distinguishing advantage of $\mathcal{Z}$ between c and $\tilde{c}$ is at most $\mathbf{Adv}_{\mathcal{R}_{\mathrm{SEC}}}^{\mathrm{SEC},\mathsf{AE}}$. Assuming that there are q_{C} C sub-sessions, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_1,\mathbf{G}_2} \leq q_{\mathrm{C}} \cdot \mathbf{Adv}_{\mathcal{R}_{\mathrm{SEC}}}^{\mathrm{SEC},\mathsf{AE}},$$

which is a negligible function of κ.

In $\mathbf{G}_3$, in the case of $(*)$, replace the real execution of Π with its simulation by $\mathsf{SIM}_{\mathrm{AKE}}$. Concretely,

(1) When $\mathcal{Z}$ inputs $(\mathsf{SvrSession}, sid, ssid)$ to S, send $(\mathsf{SvrSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM}_{\mathrm{AKE}}$. Also, if this is the first $\mathsf{SvrSession}$ message for $ssid$, record $(ssid, \mathsf{S}, \mathsf{C})$ and mark it fresh;

(2) When C's OPRF sub-session is completed and $\mathcal{A}$ sends c^* to C, send $(\mathsf{CltSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM}_{\mathrm{AKE}}$. Also, record $(ssid, \mathsf{C}, \mathsf{S})$ and mark it fresh;

(3) When $\mathcal{A}$ sends (Compromise, sid) to S, send (Compromise, sid, S) to $\mathsf{SIM}_{\mathrm{AKE}}$; when $F_{\mathsf{S}}(\mathsf{pw})$ is defined, send (Compromise, sid, C) to $\mathsf{SIM}_{\mathrm{AKE}}$;

(4) On (Impersonate, sid, $ssid$, P) from $\mathsf{SIM}_{\mathrm{AKE}}$, if there was a (Compromise, sid, P) message, mark ($ssid$, C, P) compromised;

(5) While $\mathsf{SIM}_{\mathrm{AKE}}$ simulates the execution of Π, pass messages between it and $\mathcal{A}$;

(6) On (NewKey, sid, $ssid$, P, SK^*) from $\mathsf{SIM}_{\mathrm{AKE}}$ where $|SK^*| = \kappa$, if there is a record ($ssid$, P, P′) not marked completed, do:

- If P = S, ignore this message if there is no (sid, $ssid$, C) session marked completed.
- Else if the record is marked compromised, or either P or P′ is corrupted, set $SK := SK^*$.
- Else if the record is marked fresh, a (sid, $ssid$, SK') tuple was sent to P′, and at that time there was a record ($ssid$, P′, P) marked fresh, set $SK := SK'$.
- Else pick $SK \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$.

Finally, mark ($ssid$, P, P′) completed and send (sid, $ssid$, SK) to P.

We construct an environment $\mathcal{Z}_{\mathrm{AKE}}$ against Π as follows:

(1) When $\mathcal{Z}$ inputs (SvrSession, sid, $ssid$) to S, send (SvrSession, sid, $ssid$) to S;

(2) When C's OPRF sub-session is completed and $\mathcal{A}$ sends c^* to C, input (CltSession, sid, $ssid$) to C;

(3) In the execution of Π, $\mathcal{Z}_{\mathrm{AKE}}$ behaves as $\mathcal{Z}$, with the "dummy" adversary as a sub-procedure.

(4) $\mathcal{Z}_{\mathrm{AKE}}$ copies $\mathcal{Z}$'s output. (Intuitively, $\mathcal{Z}_{\mathrm{AKE}}$ guesses "real world" if $\mathcal{Z}$ guesses "$\mathbf{G}_2$," and "simulated world" if $\mathcal{Z}$ guesses "$\mathbf{G}_3$.")

Note that p_c (resp. p_s) is random in $\mathcal{Z}$'s view unless and until a $(\mathsf{Compromise}, sid, \mathsf{C})$ (resp. $(\mathsf{Compromise}, sid, \mathsf{S})$) message is sent to $\mathsf{SIM}_{\mathrm{AKE}}$, which is consistent with the setting of AKE simulation. Therefore,

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_2,\mathbf{G}_3} \leq \mathbf{Dist}_{\mathcal{Z}_{\mathrm{AKE}}}^{\Pi,\{\mathcal{F}_{\mathrm{AKE-KCI}},\mathsf{SIM}_{\mathrm{AKE}}\}},$$

which is a negligible function of κ.

Note that in $\mathbf{G}_3$, C outputs $(\mathsf{abort}, sid, ssid)$ if and only if $\mathsf{AuthDec}_{\mathsf{rw}'}(c^*) = \perp$ (where $\mathsf{rw}' = F_{\mathsf{S}^*}(\mathsf{pw}')$). In the following three games we gradually change this condition.

In $\mathbf{G}_4$, in the case that $c^* = c$, $\mathsf{S}^* = \mathsf{S}$ but $\mathsf{pw}' \neq \mathsf{pw}$, C outputs $(\mathsf{abort}, sid, ssid)$.

$\mathcal{Z}$'s views in $\mathbf{G}_3$ and $\mathbf{G}_4$ are identical unless $c^* = c$, $\mathsf{S}^* = \mathsf{S}$, $\mathsf{pw}' \neq \mathsf{pw}$, but $\mathsf{AuthDec}_{\mathsf{rw}'}(c) \neq \perp$ (in which case C outputs $(sid, ssid, SK)$ in $\mathbf{G}_3$ and $(\mathsf{abort}, sid, ssid)$ in $\mathbf{G}_4$). Since $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_c, P_c, P_s)$, we have that $\mathsf{AuthDec}_{\mathsf{rw}}(c) \neq \perp$. But rw' and rw are independent random strings in $\{0,1\}^{2\kappa}$, so by a reduction $\mathcal{R}_{\mathrm{RBST1}}$ to the random-key robustness of AE, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_3,\mathbf{G}_4} \leq q_{\mathrm{C}} \cdot \mathbf{Adv}_{\mathcal{R}_{\mathrm{RBST1}}}^{\mathrm{RBST},\mathsf{AE}},$$

which is a negligible function of κ.

In $\mathbf{G}_5$, in the case that $\neg(c^* = c \wedge \mathsf{S}^* = \mathsf{S})$, output halt and abort if there are $x_1 \neq x_2$ such that $\mathcal{A}$ queries both $y_1 = F_{\mathsf{S}^*}(x_1)$ and $y_2 = F_{\mathsf{S}^*}(x_2)$ (call "$\mathcal{A}$ queries $F_{\mathsf{S}^*}(x)$" if (i) $\mathcal{A}$ sends $(\mathsf{Eval}, sid, ssid, x)$ and then $(\mathsf{RcvComplete}, sid, ssid, \mathcal{A}, \mathsf{S}^*)$ to $\mathcal{F}_{\mathrm{OPRF}}$, or (ii) $\mathsf{S}^* = \mathsf{S}$, $x = \mathsf{pw}$, and $\mathcal{A}$ sends $(\mathsf{OfflineEval}, sid, \mathsf{S}, \mathsf{pw})$ to $\mathcal{F}_{\mathrm{OPRF}}$ when S is corrupted or marked $\mathsf{compromised}$), and $\mathsf{AuthDec}_{y_1}(c^*), \mathsf{AuthDec}_{y_2}(c^*) \neq \perp$.

Note that y_1 and y_2 are independent random strings in $\{0,1\}^{2\kappa}$. By a reduction $\mathcal{R}_{\mathrm{RBST2}}$ to the random-key robustness of AE, for a single C sub-session and two fixed y_1 and y_2, $\Pr[\mathsf{AuthDec}_{y_1}(c^*), \mathsf{AuthDec}_{y_2}(c^*) \neq \perp] \leq \mathbf{Adv}_{\mathcal{R}_{\mathrm{RBST2}}}^{\mathrm{RBST},\mathsf{AE}}$. Assuming that $\mathcal{A}$ sends q_{F} Eval and

$\mathsf{OfflineEval}$ messages to $\mathcal{F}_{\mathrm{OPRF}}$ in total, by a polynomial reduction, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_4,\mathbf{G}_5} \leq \Pr[\mathsf{halt}] \leq q_{\mathrm{F}}^2 q_{\mathrm{C}} \cdot \mathbf{Adv}_{\mathcal{R}_{\mathrm{RBST2}}}^{\mathrm{RBST},\mathsf{AE}},$$

which is a negligible function of κ.

In $\mathbf{G}_6$, in the case that $\neg(c^* = c \wedge \mathsf{S}^* = \mathsf{S})$, C outputs $(\mathsf{abort}, sid, ssid)$ if $\mathcal{A}$ does not query $F_{\mathsf{S}^*}(\mathsf{pw}')$.

$\mathcal{Z}$'s views in $\mathbf{G}_5$ and $\mathbf{G}_6$ are identical unless $\mathcal{A}$ does not query $\mathsf{rw}' = F_{\mathsf{S}^*}(\mathsf{pw}')$ but $\mathsf{AuthDec}_{\mathsf{rw}'}(c^*) = \perp$ (in which case C outputs $(sid, ssid, SK)$ in $\mathbf{G}_5$ and $(\mathsf{abort}, sid, ssid)$ in $\mathbf{G}_6$). Since $\mathcal{A}$ does not query $F_{\mathsf{S}^*}(\mathsf{pw}')$, rw' is a random string in $\{0,1\}^{2\kappa}$ in $\mathcal{Z}$'s view. $\mathcal{Z}$ additionally learns $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_c, P_c, P_s)$, but if $\mathsf{S}^* \neq \mathsf{S}$, rw is independent of rw'; if $\mathsf{S}^* = \mathsf{S}$, $\mathcal{A}$'s message is restricted to $c^* \neq c$. By a reduction $\mathcal{R}_{\mathrm{AUTH}}$ to the authenticity of AE, for a single C sub-session, the probability that $\mathcal{A}$ sends c^* such that $\mathsf{AuthDec}_{\mathsf{rw}'}(c^*) = \perp$ is at most $\mathbf{Adv}_{\mathcal{R}_{\mathrm{AUTH}}}^{\mathrm{AUTH},\mathsf{AE}}$. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_5,\mathbf{G}_6} \leq q_{\mathrm{C}} \cdot \mathbf{Adv}_{\mathcal{R}_{\mathrm{AUTH}}}^{\mathrm{AUTH},\mathsf{AE}},$$

which is a negligible function of κ.

Note that in $\mathbf{G}_6$, $\mathsf{rw}' = F_{\mathsf{S}^*}(\mathsf{pw}')$ is defined no matter $\mathcal{A}$ queries it or not. In $\mathbf{G}_7$, leave rw' undefined unless and until $\mathcal{A}$ queries $F_{\mathsf{S}^*}(\mathsf{pw}')$.

If $\mathcal{A}$ does not query $F_{\mathsf{S}^*}(\mathsf{pw}')$, rw' is a random string in $\{0,1\}^{2\kappa}$ in $\mathcal{Z}$'s view. There are q_{C} rw''s. Therefore, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_6,\mathbf{G}_7} \leq \frac{q_{\mathrm{C}}}{2^{2\kappa}},$$

which is a negligible function of κ.

$\mathbf{G}_8$ is the simulated world. We can see that the change from $\mathbf{G}_7$ to $\mathbf{G}_8$ is merely conceptual,

with the game challenger split into the saPAKE functionality $\mathcal{F}$ and the simulator SIM. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathbf{G}_7,\mathbf{G}_8} = 0.$$

Summing up all results above, we conclude that $\mathcal{Z}$'s distinguishing advantage between the real world and the simulated world is a negligible function of κ. This completes the proof.

□

5.6 OPAQUE: A Strong Asymmetric PAKE Instantiation

Figure 5.11 shows OPAQUE, a concrete instantiation of the generic OPRF+AKE protocol from Figure 5.8.

The OPRF is instantiated with the 2HashDH protocol, while the AKE protocol can be instantiated with any 3-message UC-secure AKE-KCI with the server as the last-to-complete party; in Figure 5.11 this is illustrated with the 3-message HMQV [56]. Note that the two messages of 2HashDH and two of the three messages from HMQV (or a similar protocol) run "in parallel," hence obtaining a 3-message saPAKE.

By Theorem 9 on the security of the generic OPRF+AKE construction, by Theorem 7 on the security of 2HashDH, and by security of HMQV, we get that protocol OPAQUE realizes functionality $\mathcal{F}_{\text{saPAKE}}$, hence it is a provably-secure strong aPAKE, under the OMDH assumption in ROM.

Public Parameters and Components

- Group $\mathbb{G}$ of prime order m ($|m| = 2\kappa$) and generator g ($\mathbb{G}^*$ denotes $\mathbb{G} \setminus \{1_\mathbb{G}\}$).
- Hash functions $H(\cdot,\cdot)$ and $H'(\cdot)$ with ranges $\{0,1\}^{2\kappa}$ and $\mathbb{G}$, respectively.
- PRF $f(\cdot)$ with range $\{0,1\}^{2\kappa}$.
- Key-committing authenticated encryption scheme $\mathsf{AE} = (\mathsf{KeyGen}, \mathsf{AuthEnc}, \mathsf{AuthDec})$ where $\mathsf{KeyGen}(1^\kappa)$ picks a key uniformly at random from $\{0,1\}^{2\kappa}$.
- Key exchange formula KE defined below.

Password Registration

1. On $(\mathsf{StorePwdFile}, sid, \mathsf{C}, \mathsf{pw})$, S picks $k \leftarrow_\mathrm{R} \mathbb{Z}_m$ and computes $\mathsf{rw} := F_k(\mathsf{pw}) := H(\mathsf{pw}, H'(\mathsf{pw})^k)$; picks $p_s, p_c \leftarrow_\mathrm{R} \mathbb{Z}_m$ and computes $P_s := g^{p_s}$, $P_c := g^{p_c}$, and $c \leftarrow \mathsf{AuthEnc}_\mathsf{rw}(p_c, P_c, P_s)$; and records $\mathsf{file}[sid] := \langle k, p_s, P_s, P_c, c\rangle$.

Login

1. On $(\mathsf{CltSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$, C picks $r, x_c \leftarrow_\mathrm{R} \mathbb{Z}_m$; computes $a := H'(\mathsf{pw})^r$ and $X_c := g^{x_c}$; and sends a and X_c to S.
2. On $(\mathsf{SvrSession}, sid, ssid)$ and a from C, S checks if $a \in \mathbb{G}^*$, and if not, it outputs $(\mathsf{abort}, sid, ssid)$ and halts. Else it retrieves $\mathsf{file}[sid] = \langle k, p_s, P_s, P_c, c\rangle$; picks $x_s \leftarrow_\mathrm{R} \mathbb{Z}_m$ and computes $b := a^k$ and $X_s := g^{x_s}$; computes $K_s := \mathsf{KE}(p_s, x_s, P_c, X_c)$ and $SK_s := f_{K_s}(0)$; and sends b, X_s and c to C.
3. On b, X_s and c from S, C checks if $b \in \mathbb{G}^*$, and if not, it outputs $(\mathsf{abort}, sid, ssid)$ and halts. Else it computes $\mathsf{rw}' := H(\mathsf{pw}', b^{1/r})$ and $\mathsf{AuthDec}_{\mathsf{rw}'}(c)$.
   If the result is $\perp$, C outputs $(\mathsf{abort}, sid, ssid)$ and halts.
   Else C parses $(p_c^*, P_c^*, P_s^*) := \mathsf{AuthDec}_{\mathsf{rw}'}(c)$; computes $K_c := \mathsf{KE}(p_c^*, x_c, P_s^*, X_s)$ and $SK_c := f_{K_c}(0)$; sends $t := f_{K_c}(1)$ to S and outputs $(sid, ssid, SK_c)$.
4. On t from C, S checks if $t = f_{K_s}(1)$. If so, S outputs $(sid, ssid, SK_s)$. Else it outputs $(\mathsf{abort}, sid, ssid)$.

Key exchange formula KE with HMQV instantiation (if any of $X_c, P_c, X_s, P_s \notin \mathbb{G}^*$, the receiving party outputs $(\mathsf{abort}, sid, ssid)$ and halts)

$$\begin{aligned} \text{For } \mathsf{S}: \quad \mathsf{KE}(p_s, x_s, P_c, X_c) &= H''((X_c P_c^{e_u})^{x_s + e_s p_s}) \\ \text{For } \mathsf{C}: \quad \mathsf{KE}(p_c, x_c, P_s, X_s) &= H''((X_s P_s^{e_s})^{x_c + e_u p_c}) \end{aligned}$$

where $e_u = H(X_c, \mathsf{S}) \bmod m$, $e_s = H(X_s, \mathsf{C}) \bmod m$.

Figure 5.11: Protocol OPAQUE

5.6.1 Protocol Details and Properties

We expand on the specification of OPAQUE and the protocol's properties.

- *Password registration:* Password registration is the only part of the protocol assumed to run over secure channels where parties can authenticate each other. We note that while OPAQUE is presented with the server doing all the registration operations, in practice one may want to avoid that. Instead, we can let the server choose an OPRF key k and the client choose the password pw, and then run the OPRF protocol between the client and the server so only the client learns its secrets (pw, rw and p_c) and only S learns p_s.

  A problem arises with this approach if the server's policy is to check the client's password for compliance with some rules. A possible workaround is to adapt techniques from [54] that present zero-knowledge proofs for proving compliance without disclosing the password.

- *Authenticated encryption:* As specified in Section 5.5.2, the scheme AE used in the protocol needs to satisfy random-key robustness defined there. In practice, using an encrypt-then-MAC scheme with HMAC-256 (or larger) as the MAC provides this property (if a scheme does not have this property then adding on top of it such a HMAC computed on the scheme's ciphertext will ensure this property).

- *Key exchange:* The generic AKE representation via the KE formula applies to any protocol whose session key is computed as a function of the long-term private-public key pair of each party and ephemeral session-specific private-public values. These values are represented as (p_s, P_s, x_s, X_s) for the server and (p_c, P_c, x_c, X_c) for the client. We note that while more general key-exchange protocols can be used with OPAQUE, this representation applies to many such protocols and, in particular, to HMQV [56] which we use here as our main instantiation.

- *Explicit mutual authentication:* The protocol as illustrated only provides explicit client authentication. With an additional message the protocol achieves mutual authentication by simply adding the value $f_{K_s}(2)$ to the server's message. The client verifies that the value received from the server is computed correctly and if not it aborts.

- *Use of HMQV:* Recall that the security of OPAQUE depends on the KE protocol being AKE-secure in the UC framework with the additional KCI property; namely, it should realize the UC AKE-KCI functionality from Figure 5.7. As argued in Section 5.5.1, HMQV indeed realizes this functionality (under the CDH assumption in ROM), hence it is appropriate for use in OPAQUE.

- *Forward secrecy:* This property (or lack of it) is inherited by OPAQUE from the key exchange component KE. In the case of (3-message) HMQV, it provides full forward secrecy, namely, against arbitrary active attacks [56]. *One cannot overstate the importance of forward secrecy in password protocols: it guarantees that past session keys remain secure upon the compromise of a client's password (or server's information).*

- *Client iterated hashing:* OPAQUE can be strengthened by increasing the cost of a dictionary attack in case of server compromise. This is done by changing the computation of rw to $\mathsf{rw} = H^n(F_k(\mathsf{pw}))$, that is, the client applies n iterations of the function $H(\cdot)$ on top of the result of the OPRF value $F_k(\mathsf{pw})$. In practice, the iterations H^n would be replaced with one of the standard password-based KDFs, such as PBKDF2 [50] or PM99 [66]. This forces an attacker that compromises the password file at the server to compute for *each* candidate password pw' the function $F_k(\mathsf{pw}')$ as well as the additional n hash iterations. Note that n does not need to be remembered by the client; it can be sent from S to C in the server's message. Furthermore, one can follow Boyen's design and apply the probabilistic Halting KDF

function [17] as used in [18] so that the iterations count is hidden from the attacker and even from the server.

- *Performance:* OPAQUE takes three messages; 1 exponentiation for S, 2 and a hashing-into-$\mathbb{G}$ for C, plus the cost of KE. With HMQV, the latter cost is 1 offline fixed-base exponentiation and 1 multi-exponentiation (at the cost of 1.16 regular exponentiations) per party (about three exponentiations in total for the server and four for the client). All exponentiations are in regular DH groups, hence accommodating the fastest elliptic curves (e.g., no pairings). It is common in PAKE protocols to count number of group elements transmitted between the parties. In OPAQUE, C sends 2 while S sends 3 (one, P_c, can be omitted at the cost of 1 fixed-based exponentiation at the client).

- *Performance comparison:* The introduction presents background on OPAQUE and other password protocols. Here we provide a comparison with the more efficient among these protocols, particularly those that are being, or have been, considered for standardization. Clearly, OPAQUE is superior security-wise as the only one not subject to pre-computation attacks, but it also fares well in terms of performance.

  AugPAKE [69] is computationally very efficient with only 2.17 exponentiations per party; however, it uses 4 messages and does not provide forward secrecy. In addition, the protocol has only been analyzed as a PAKE protocol, not aPAKE.

  Another proposed aPAKE protocol, SPAKE2+ [6, 29], uses two messages only and 3 multi-exponentiations (or about 3.5 exponentiations) per party which is similar to OPAQUE cost. The security of the protocol has only been informally argued in [29] and to the best of our knowledge no formal analysis has appeared.

  We also mention SRP which has been included in TLS ciphersuites in the past but is considered outdated as it does not have an instantiation that works over elliptic curves (the protocol is defined over rings and uses both addition and multiplication). Its implementations over RSA moduli is therefore less efficient than those over elliptic

curve; it also takes 4 messages.

We also mention two very recent protocols that have been formally analyzed as aPAKE protocols but, as the rest, are vulnerable to pre-computation. The protocol VTBPEKE in [65] uses 3 messages and 4 exponentiations per party and was proven secure in the non-UC aPAKE model of [14], while [49] shows a *simultaneous* one-round protocol that they prove secure in the UC aPAKE model of [40] augmented with adaptive security. The protocol works over bilinear groups and its computational cost includes 4 exponentiations and 3 pairing per party.

We note that all of the above protocols require an initial message from server to client in order to transmit salt, which may result in one or two added messages to the above message counts (except for VTBPEKE which already includes the salt transmission in its 3 messages). Also, all these protocols, like OPAQUE, work in ROM.

- *Threshold implementation:* We comment on a simple extension of OPAQUE that can be very valuable in large deployments, namely, the ability to implement the OPRF phase as a Threshold OPRF, as introduced in Chapter 3. In this case, an attacker needs to break into a threshold of servers to be able to impersonate the servers to the client or to run an offline dictionary attack. Such an implementation requires no client-side changes, i.e., the client does not need to know if the system is implemented with one or multiple servers.

- *OPAQUE as a general secret retrieval mechanism:* An important feature of OPAQUE is that it can serve not only as an aPAKE protocol but more generally as a means for retrieving a secret or credential from a server (such a secret is protected under ciphertext c stored at the server). In this functionality, OPAQUE acts as a 1-out-of-1 implementation of the TOPPSS protocol in Chapter 3. The retrieved secret can be used to protect information such as a bitcoin wallet, serve as a client-controlled encryption key for a backup or other information repository (e.g., a password manager), used as

an authentication or signing key, and more. This offers a far more secure alternative to the practice of deriving low-entropy secrets directly from a client's password.

5.6.2 OPAQUE and TLS: Client Authentication and Hedging Against PKI Failures

As discussed earlier, OPAQUE offers a much more secure alternative to password-authenticated key exchange than the current practice of transmitting passwords over TLS. Yet, OPAQUE (as any other aPAKE) still requires additional mechanisms for negotiating cryptographic parameters (such as crypto algorithms) and for establishing the means needed to encrypt and authenticate communications using the keys generated by OPAQUE. Thus, it is natural to compose OPAQUE with the TLS protocol to offer strong password security while leveraging the standardized negotiation and record-layer security of TLS. Moreover, TLS can offer an initial server-authenticated channel to protect the privacy of account information, such as username, transmitted between client and server. Here we discuss possible protocols for composing OPAQUE and TLS. We consider TLS 1.3 [67] as the upcoming and more secure version of TLS, although some of the mechanisms can be implemented via prior versions of TLS.

The simplest TLS-OPAQUE combination is one where C's private key p_c stored by OPAQUE at the server is used as a signature key for TLS client authentication. In this case, the OPAQUE-extended handshake protocol includes the following sequential steps (for a total of 5 messages): (1) a 1-RTT run of TLS 1.3 handshake protocol that produces a session key authenticated by S's TLS certificate; (2) the two OPAQUE messages exchanged between client and server excluding the KE values g^x and g^y (these were already exchanged as part of the TLS 1-RTT run); (3) TLS 1.3 client authentication using the client's private signature key p_c retrieved from the server in step (2).

These steps result in mutual authentication where server's authentication is accomplished based on a TLS certificate. The client can either trust such a certificate or it can verify equality of the certificate's public key against P_s as retrieved by OPAQUE. In case of a mismatch the client can request a signature of S using P_s which is computed on the TLS transcript[4]. In the latter case, the protocol does not rely on PKI certificates except for protecting account information. *In all cases, the security of passwords and password authentication does not rely on PKI but on OPAQUE only.*

Variants of the above protocol include the use of a TLS 1.3 0-RTT exchange for sending the first OPAQUE message (including protected account information) in which case steps (1) and (2) are executed concurrently for a total of two messages (two flights in TLS jargon). This variant, while more efficient, relies on 0-RTT which is available only to clients and servers that have previously shared a key (negotiated in a previous handshake). A 0-RTT variant independent of pre-shared keys and based instead on a server's public key is possible (e.g., [58]) but it is not standardized by TLS 1.3. Finally, if protecting the secrecy of client's account information is not considered necessary then steps (1) and (2) can run concurrently (without using the 0-RTT protocol); in this case server's authentication is based on the server's stored key P_s. This setting also allows for a maximally efficient protocol using HMQV as illustrated in Figure 5.11 (with additional key derivation and record layer processing based on TLS).

We note that the security of the above variants and composition rely on the modularity of OPAQUE that can compose the OPRF steps with arbitrary key-exchange protocols (with KCI security). We remark that the security of TLS 1.3 has been analyzed in multiple works (cf. [34, 32, 31, 15, 36, 59] with client authentication via exported authentication (or "post-handshake authentication") studied in [57].

[4]Such additional server authentication and the client authentication in step (3) can be implemented using TLS exported authenticators as defined in [72] (client authentication in this case corresponds to post-handshake authentication in [67]).

Chapter 6

Conclusion and Future Work

This study addresses two major topics in password authentication in the client-server setting, namely Password-Protected Secret Sharing (PPSS) and asymmetric Password-Authenticated Key Exchange (aPAKE). For PPSS, we present TOPPSS, which is a highly-efficient PPSS protocol proven secure in the Universally Composable (UC) framework. For aPAKE, we first present a round-reduced protocol based on the Ω-method compiler of [40]. We then propose strong aPAKE (saPAKE), which eliminates the so-called pre-computation attack, hence achieving the motivation behind the concept of aPAKE. We present OPAQUE, which is the first saPAKE protocol in the literature, with efficiency comparable to normal aPAKE protocols, and also prove it secure in the UC framework.

Future work. Below we list several specific problems emerging from this study, which create possible avenues for further exploration.

- *Threshold (a)PAKE:* There has been relatively little work on *Threshold PAKE (T-PAKE)* since it was proposed in [63]. [43], which shows the close relationship between PPSS and T-PAKE by constructing a generic compiler from the former to the latter,

has the security proof in the game-based model. Since we have a UC notion of PPSS in this study, a natural question is: Can we prove the security of this compiler in the UC framework?

Furthermore, to the best of our knowledge, there is no threshold (strong) aPAKE notion in the literature (in either the game-based model or the UC framework), yet such protocols have much practical value. Thus, "thresholdizing" efficient UC aPAKE protocols is an interesting open problem.

- *Strong aPAKE under weaker assumptions:* A drawback of the OPAQUE protocol is that its security relies on the somewhat non-standard and interactive OMDH assumption. It is natural to ask whether strong aPAKE protocols can be constructed under weaker assumptions such as DDH, with similar computational cost. Since saPAKE is a new research topic and the only existing constructions are the ones presented in this study, there may be much space for improvement.

- *Two-Factor Authentication (TFA):* Two-factor authentication (TFA), in which a client authenticates with a server using a password and a piece of additional information (e.g., a PIN) sent from a supplementary device, has been widely used in practice; however, there is little work on the formal security models and security arguments of such protocols in the cryptographic literature. The first such formal treatment [70] defines both online security and offline security (upon server compromise) in the game-based model, yet the protocols in [70] rely on secure channels.

  Similar to the PAKE problem, a natural question is whether we can eliminate the usage of secure channels, hence achieving the same security level in the password-only setting. If so, the resulting protocol can be viewed as a strengthening of aPAKE, with offline security and online security both increasing by a factor of 2^t, where t is the length of the PIN. Specifically, the running time of a successful offline dictionary attack increases 2^t times, and the probability of a successful online guessing attack decreases 2^t times.

Bibliography

[1] CFRG, Crypto Forum Research Group, `https://datatracker.ietf.org/rg/cfrg/documents/`.

[2] M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In *Topics in Cryptology – CT-RSA 2008*, pages 335–351. Springer, 2008.

[3] M. Abdalla, O. Chevassut, P.-A. Fouque, and D. Pointcheval. A simple threshold authenticated key exchange from short secrets. In *Advances in Cryptology – ASIACRYPT 2005*, pages 566–584. Springer, 2005.

[4] M. Abdalla, O. Chevassut, and D. Pointcheval. One-time verifier-based encrypted key exchange. In *Public-Key Cryptography – PKC 2005*, pages 47–64. Springer, 2005.

[5] M. Abdalla, M. Cornejo, A. Nitulescu, and D. Pointcheval. Robust password-protected secret sharing. In *Computer Security – ESORICS 2016*, pages 61–79. Springer, 2016.

[6] M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *Topics in Cryptology – CT-RSA 2005*, pages 191–208. Springer, 2005.

[7] M. Abe and M. Ohkubo. A framework for universally composable non-committing blind signatures. In *Advances in Cryptology – ASIACRYPT 2009*, pages 435–450. Springer, 2009.

[8] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *ACM Conference on Computer and Communications Security - CCS 2011*, pages 433–444. ACM, 2011.

[9] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3), 2003.

[10] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – EUROCRYPT 2000*, pages 139–155. Springer, 2000.

[11] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security – CCS 1993*, pages 62–73. ACM, 1993.

[12] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Computer Society Symposium on Research in Security and Privacy – S&P 1992*, pages 72–84. IEEE, 1992.

[13] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM Conference on Computer and Communications Security – CCS 1993*, pages 244–250. ACM, 1993.

[14] F. Benhamouda and D. Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. *IACR Cryptology ePrint Archive*, 2013:833, 2013.

[15] K. Bhargavan, B. Blanchet, and N. Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *IEEE Computer Society Symposium on Research in Security and Privacy – S&P 2017*, pages 483–502. IEEE, 2017.

[16] O. Blazy, C. Chevalier, and D. Vergnaud. Mitigating server breaches in password-based authentication: Secure and efficient solutions. In *Topics in Cryptology – CT-RSA 2016*, pages 3–18. Springer, 2016.

[17] X. Boyen. Halting password puzzles. In *USENIX Security Symposium – SECURITY 2007*, pages 119–134. The USENIX Association, 2007.

[18] X. Boyen. HPAKE: Password authentication secure against cross-site user impersonation. In *Cryptology and Network Security – CANS 2009*, pages 279–298. Springer, 2009.

[19] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Advances in Cryptology – EUROCRYPT 2000*, pages 156–171. Springer, 2000.

[20] J. Brainard, A. Juels, B. Kaliski, and M. Szydlo. Nightingale: A new two-server approach for authentication with short secrets. In *USENIX Security Symposium - SECURITY 2003*, pages 201–213. The USENIX Association, 2003.

[21] J. Camenisch, N. Casati, T. Gross, and V. Shoup. Credential authenticated identification and key exchange. In *Advances in Cryptology – CRYPTO 2010*, pages 255–276. Springer, 2010.

[22] J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In *Advances in Cryptology – CRYPTO 2014*, pages 256–275. Springer, 2014.

[23] J. Camenisch, A. Lehmann, and G. Neven. Optimal distributed password verification. In *ACM Conference on Computer and Communications Security - CCS 2015*, pages 182–194. ACM, 2015.

[24] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science – FOCS 2001*, pages 136–145. IEEE, 2001.

[25] R. Canetti, A. Cohen, and Y. Lindell. A simpler variant of universally composable security for standard multiparty computation. In *Advances in Cryptology – EUROCRYPT 2015*, pages 3–22. Springer, 2015.

[26] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology – EUROCRYPT 2005*, pages 404–421. Springer, 2005.

[27] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology – EUROCRYPT 2001*, pages 453–474. Springer, 2001.

[28] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Advances in Cryptology – EUROCRYPT 2002*, pages 337–351. Springer, 2002.

[29] D. Cash, E. Kiltz, and V. Shoup. The twin Diffie-Hellman problem and applications. In *Advances in Cryptology – EUROCRYPT 2008*, pages 127–145. Springer, 2008.

[30] S. Contini. Method to protect passwords in databases for web applications. *IACR Cryptology ePrint Archive*, 2015:387, 2015.

[31] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *ACM Conference on Computer and Communications Security CCS 2017*, pages 1773–1788. ACM, 2017.

[32] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe. Automated verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *IEEE Computer Society Symposium on Research in Security and Privacy – S&P 2016*, pages 470–485. IEEE, 2016.

[33] M. Di Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. In *Advances in Cryptology – EUROCRYPT 2003*, pages 507–523. Springer, 2003.

[34] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In *ACM Conference on Computer and Communications Security - CCS 2015*, pages 1197–1210. ACM, 2015.

[35] P. Farshim, C. Orlandi, and R. Rosie. Security of symmetric primitives under incorrect usage of keys. *IACR Transactions on Symmetric Cryptology*, 2017(1):449–473, 2017.

[36] M. Fischlin and F. Güther. Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In *IEEE Computer Society Symposium on Research in Security and Privacy – S&P 2017*, pages 60–75. IEEE, 2017.

[37] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography – TCC 2005*, pages 303–324. Springer, 2005.

[38] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[39] R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In *Advances in Cryptology – EUROCRYPT 2003*, pages 524–543. Springer, 2003.

[40] C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *Advances in Cryptology – CRYPTO 2006*, pages 142–159. Springer, 2006.

[41] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. In *ACM Conference on Computer and Communications Security – CCS 1998*, pages 122–131. ACM, 1998.

[42] J. Y. Hwang, S. Jarecki, T. Kwon, J. Lee, J. S. Shin, and J. Xu. Round-reduced modular construction of asymmetric password-authenticated key exchange. In *Security and Cryptography for Networks – SCN 2018*, pages 485–504. Springer, 2018.

[43] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *Advances in Cryptology – ASIACRYPT 2014*, pages 233–253. Springer, 2014.

[44] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. Highly-efficient and composable password-protected secret sharing (Or: how to protect your bitcoin wallet online). In *IEEE European Symposium on Security and Privacy – EuroS&P 2016*, pages 276–291. IEEE, 2016.

[45] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In *Applied Cryptography and Network Security – ACNS 2017*, pages 39–58. Springer, 2017.

[46] S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In *Advance in Cryptology – EUROCRYPT 2018*, pages 456–486. Springer, 2018.

[47] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *Security and Cryptography for Networks – SCN 2010*, pages 418–435. Springer, 2010.

[48] C. S. Jutla and A. Roy. Relatively-sound NIZKs and password-based key-exchange. In *Public-Key Cryptography - PKC 2012*, pages 485–503. Springer, 2012.

[49] C. S. Jutla and A. Roy. Smooth NIZK arguments. In *Theory of Cryptography – TCC 2018*, pages 235–262. Springer, 2018.

[50] B. Kaliski. Requirements for password-authenticated key agreement (PAKE) schemes. Technical report, 2017.

[51] J. Katz, P. Mackenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. In *Applied Cryptology and Network Security – ACNS 2005*, pages 1–16. Springer, 2005.

[52] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Advances in Cryptology – EUROCRYPT 2001*, pages 475–494. Springer, 2001.

[53] J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. *Journal of Cryptology*, 26(4):714–743, 2013.

[54] F. Kiefer and M. Manulis. Zero-knowledge password policy checks and verifier-based PAKE. In *Computer Security – ESORICS 2014*, pages 295–312. Springer, 2014.

[55] F. Kiefer and M. Manulis. Universally composable two-server PAKE. In *Informational Security – ISC 2016*, pages 147–166. Springer, 2016.

[56] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol (extended abstract). In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. Springer, 2005.

[57] H. Krawczyk. Unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in TLS 1.3). In *ACM Conference on Computer and Communications Security - CCS 2016*, pages 1438–1450. ACM, 2016.

[58] H. Krawczyk and H. Wee. The OPTLS protocol and TLS 1.3. In *IEEE European Symposium on Security and Privacy – EuroS&P 2016*, pages 81–96. IEEE, 2016.

[59] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu. Multiple handshakes security of TLS 1.3 candidates. In *IEEE Computer Society Symposium on Research in Security and Privacy – S&P 2016*, pages 486–505. IEEE, 2016.

[60] J. Lopez, R. Oppliger, and G. Purnel. Why have public key infrastructures failed so far? *Internet Research*, 15(5), 2005.

[61] P. Mackenzie. More efficient password-authenticated key exchange. In *Topics in Cryptology – CT-RSA 2001*, pages 361–377. Springer, 2001.

[62] P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. In *Advances in Cryptology – ASIACRYPT 2000*, pages 599–613. Springer, 2000.

[63] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *Advances in Cryptology – CRYPTO 2002*, pages 385–400. Springer, 2002.

[64] M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In *Advances in Cryptology – EUROCRYPT 1999*, pages 327–346. Springer, 1999.

[65] D. Pointcheval and G. Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In *ACM Asia Conference on Computer and Communications Security – AsiaCCS 2017*, pages 301–312. ACM, 2017.

[66] N. Provos and D. Mazieres. A future-adaptable password scheme. In *USENIX Annual Technical Conference – FREENIX 1999*, pages 81–91. The USENIX Association, 1999.

[67] E. Rescorla. The Transport Layer Security (TLS) protocol version 1.3 (draft 24), `https://www.ietf.org/id/draft-ietf-tls-tls13-24.txt`, Feb 2018.

[68] J. Schmidt. PKCS# 5: Password-based cryptography specification version 2.0. Technical report, 2000.

[69] S. Shin, K. Kobara, and H. Imai. Security proof of AugPAKE. *IACR Cryptology ePrint Archive*, 2010:334, 2010.

[70] M. Shirvanian, S. Jarecki, N. Saxena, and N. Nathan. Two-factor authentication resilient to server compromise using mix-bandwidth devices. In *Network and Distributed System Security Symposium – NDSS 2014*, 2014.

[71] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology – EUROCRYPT 1997*, pages 256–266. Springer, 1997.

[72] N. Sullivan. Exported authenticators in TLS (draft 4), `https://tools.ietf.org/html/draft-ietf-tls-exported-authenticator-04`.

[73] M. Szydlo and B. S. Kaliski. Proofs for two-server password authentication. In *Topics in Cryptology – CT-RSA 2005*, pages 227–244. Springer, 2005.

[74] D. Wikström. Universally composable DKG with linear number of exponentiations. In *Security in Communication Networks – SCN 2004*, pages 263–277. Springer, 2004.

[75] X. Yi, F. Hao, L. Chen, and J. K. Liu. Practical threshold password-authenticated secret sharing protocol. In *Computer Security – ESORICS 2015*, pages 347–365. Springer, 2015.

Appendix A

Hardness of the (Gap) Threshold OMDH Assumption in the Generic Group Model

We show the hardness of the (Gap) T-OMDH problem defined in Section 3.3.4 in the generic group model. First in Theorem 11 we argue the $t' = 0$ case of the non-gap version of this problem, then in Theorem 12 we extend it to general t', and in Theorem 13 we explain how the proof is adapted to the case of a gap group. To the best of our knowledge, this is also the first analysis of generic group hardness of the (Gap) OMDH assumption.

Equivalence of (N, Q)-T-OMDH and $(Q+1, Q)$-T-OMDH. Note that the point which Bellare *et al.* [9] made about the One-More RSA assumption, namely that the (N, Q)-One-More problem for any $N > Q$ is equivalent to the (N, Q)-One-More problem for $N = Q + 1$, holds for the (Gap) T-OMDH problem as well, and it is a simple observation in this case because we specify the T-OMDH assumptions only in the context of a group of a known prime order. This implies in particular that in Theorems 11, 12, and 13 below it suffices to

consider the case $N = Q + 1$. The theorem below is stated (and argued) for the non-gap version of the T-OMDH assumption, but the corresponding fact holds also in the case of the gap version of this assumption.

Theorem 10. *(t', t, n, N, Q)-T-OMDH is equivalent to $(t', t, n, Q + 1, Q)$-T-OMDH.*

Proof. Given algorithm $\mathcal{A}$ against the (t', t, n, N, Q)-T-OMDH problem, we construct a reduction $\mathcal{R}$ which solves the $(t', t, n, Q + 1, Q)$-T-OMDH problem as follows:

On challenges $g_1, \ldots, g_{Q+1}$, $\mathcal{R}$ picks N sequences of random linear coefficients $\beta[i, 1], ..., \beta[i, Q + 1]$ in $\mathbb{Z}_m$ for $i = 1, \ldots, N$, sets $g'_i := g_1^{\beta[i,1]} \cdot \ldots \cdot g_{Q+1}^{\beta[i,Q+1]}$, and sends $g'_1, ..., g'_N$ as challenges set to $\mathcal{A}$. $\mathcal{R}$ passes assignment F of shares of corrupted trustees as $\mathcal{A}$ chooses them, and answers its T-OMDH oracle query of $\mathcal{A}$ by making the same query itself. Finally, if $\mathcal{A}$ outputs some $(Q + 1)$-element subset $J \subset \{1, \ldots, n\}$ and $v_j = (g'_j)^{p(0)}$ for each $j \in J$, then the $(Q + 1)$-by-$(Q + 1)$ matrix M where the k-th row is $[\beta[j_k, 1], ..., \beta[j_k, Q + 1]]$ satisfies that $[g'_{j_1}, \ldots, g'_{j_{Q+1}}] = [g_1, ..., g_{Q+1}] \cdot M^T$, where matrix multiplication stands for exponentiation, i.e., $g'_{j_k} = g_1^{M[k,1]} \cdot \ldots \cdot g_{Q+1}^{M[k,Q+1]}$. Since β's are random in $\mathbb{Z}_m$ and g_j's are random in $\mathbb{G}$, the probability that M is non-invertible is negligible (see [9]), in which case $[g_1, ..., g_{Q+1}] = [g'_{j_1}, ..., g'_{j_{Q+1}}] \cdot (M^{-1})^T$, and if $\mathcal{R}$ outputs $[z_1, ..., z_{Q+1}] = [v_{j_1}, ..., v_{j_{Q+1}}] \cdot (M^{-1})^T$ then $z_i = g_i^{p(0)}$ for each i. □

Proof of the main theorem. Before moving on to the proof of the main theorem, we briefly review the definition and some basic properties of the Hadamard product of two vectors. Recall that the Hadamard product of $\vec{a} = [a_1, \ldots, a_n]^T$ and $\vec{b} = [b_1, \ldots, b_n]^T$ is defined as

$$\vec{a} \odot \vec{b} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \odot \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ \vdots \\ a_n b_n \end{bmatrix} = \begin{bmatrix} b_1 & & \\ & \ddots & \\ & & b_n \end{bmatrix} \vec{a}.$$

From the rule above, we can see the following four properties:

(i) $\vec{a} \odot \left(\sum_{j=1}^{m} \vec{b}_j\right) = \sum_{j=1}^{m} \left(\vec{a} \odot \vec{b}_j\right)$,

(ii) $\vec{a} \odot (x\vec{b}) = x(\vec{a} \odot \vec{b})$ where x is a scalar,

(iii) If $\vec{k} \odot \vec{a} = \vec{0}$ and none of $\vec{k}$'s entries is 0, then $\vec{a} = \vec{0}$,

And by (i) and (iii) we also get:

(iv) If $\sum_{j=1}^{m} \left(\vec{k} \odot \vec{a}_j\right) = \vec{0}$ and none of $\vec{k}$'s entries is 0, then $\sum_{j=1}^{m} \vec{a}_j = \vec{0}$.

Lemma 1. *Let t be any positive integer. Then there does not exist n-dimensional vector $\vec{q}$ such that*

(1) $w \geq Qt$, and

(2) for any q_i $(i = 1, \ldots, n)$, $q_i \leq Q$,

where $w = W(\vec{q})$, and $Q = \lfloor \vec{q}/\mathcal{V}_t \rfloor + 1$.

Proof. We prove the proposition by induction on Q. If $Q = 1$, $C_t(\vec{q}) = 0$, which implies that there are at most $t - 1$ non-zero entries in $\vec{q}$; if $\vec{q}$ satisfies (2), then w is at most $t - 1$, so (1) cannot be satisfied.

Now suppose the proposition holds for $Q-1$, but not for Q, i.e. there exists $\vec{q}$ which satisfies both (1) and (2). Note that $\vec{q}$ can have at most $t - 1$ entries that are larger than or equal to Q (otherwise those t entries that are larger than or equal to Q can be decreased Q times, so $C_t(\vec{q}) \geq Q$). Let $\vec{q'}$ be $\vec{q}$ with the largest t entries decreased, and $w' = W(\vec{q'})$. Then (1) $w' = w - t \geq (Q - 1)t$, (2) for any q'_i $(i = 1, \ldots, n)$, $q'_i \leq Q - 1$ according to the above, and $C_t(\vec{q}) = Q - 2$. Therefore, $\vec{q'}$ is a counterexample for $Q - 1$, which contradicts our inductive hypothesis. □

Lemma 2. *Let t be any non-negative integer, n be any positive integer, $\vec{q}$ be an n-dimensional vector, $w = W(\vec{q})$, $Q = C_{t+1}(\vec{q}) + 1$, and $\vec{k}$ be a w-dimensional vector where there are q_i*

i's as its entries (i = 1, ..., n). Then for any w-dimensional vectors $\vec{b}_1, \ldots, \vec{b}_Q$, *the set* $V = \{\vec{k}^j \odot \vec{b}_i\}_{j\in\{0,\ldots,t\}, i\in[Q]}$ *is linearly dependent.*

Proof. Let $M : w \times Q(t+1)$ be the matrix whose columns are vectors in V. It is sufficient to show that $rank(M) < Q(t+1)$.

For $i = 1, \ldots, n$, there are q_i positions in $\vec{k}$ where the entry is i. Consider the corresponding rows in M; denote the $q_i \times Q(t+1)$ sub-matrix as M_i. Note that $rank(M_i) \leq Q$ since all its columns are multiples of its 1st, $[(t+1)+1]$-th, ..., $(Q-1)(t+1)$-th columns; therefore, for any $q_i > Q$, we can select Q rows of M_i forming matrix M_i' s.t. $rank(M_i') = rank(M_i)$. For all other q_i's, let $M_i' = M_i$. Let q_i' be the number of rows of M_i, i.e. for $q_i > Q$, let $q_i' = Q$, and for all other q_i's, let $q_i' = q_i$; then $q_i' \leq Q$ for all $i = 1, \ldots, n$. Let $w' = W(\vec{q'})$.

Now let $M' : w' \times Q(t+1)$ be the concatenation of $M_1, \ldots, M_n$. We can see that $rank(M') = rank(M)$. But $C_{t+1}(\vec{q'}) = C_{t+1}(\vec{q}) = Q - 1$. (The reason is as follows: obviously $C_{t+1}(\vec{q'}) \leq Q-1$. On the other hand, let $\vec{v}_1, \ldots, \vec{v}_{Q-1} \in \mathcal{V}_{t+1}$ such that $\vec{v}_1 + \ldots + \vec{v}_{Q-1} \leq \vec{q}$. Clearly each entry of $\vec{v}_1 + \ldots + \vec{v}_{m-1}$ is at most $Q-1$; therefore, $\vec{v}_1 + \ldots + \vec{v}_{Q-1} \leq \vec{q'}$ since $\vec{q'}$ is simply $\vec{q}$ with entries greater than Q decreased to Q. That implies $C_{t+1}(\vec{q'}) \geq Q - 1$.) According to Lemma 1, we have $w' < Q(t+1)$. So $rank(W) = rank(W') \leq w' < Q(t+1)$. □

Lemma 3. *Let t, n,* $\vec{q}$, *w, Q be the same with those in Lemma 2. Then there do not exist matrices* $A : Q \times w$, $B : w \times Q$ *and full-rank diagonal matrix* $K : w \times w$ *where there are* q_i *i's as its entries on the diagonal (*$i = 1, \ldots, n$*), s.t.* $AB = I$ *and* $AKB = \ldots = AK^tB = O$, *where I is the identity matrix and O is the zero matrix.*

Proof. Suppose $K = \begin{bmatrix} k_1 & & \\ & \ddots & \\ & & k_w \end{bmatrix}$. Let $\vec{a}_1^T, \ldots, \vec{a}_Q^T$ be the rows of A, $\vec{b}_1, \ldots, \vec{b}_Q$ be the columns of B, and $\vec{k} = [k_1, \ldots, k_w]^T$. Then $\vec{a}_1, \ldots, \vec{a}_Q, \vec{b}_1, \ldots, \vec{b}_Q, \vec{k}$ are all w-dimension column vectors, and all entries of $\vec{k}$ are non-zero.

Let $\vec{k}^j$ denote $[k_1^j, \ldots, k_w^j]^T$, and $V = \{\vec{k}^j \odot \vec{b}_i\}_{j=0,\ldots,t,i=1,\ldots,Q}$. The conditions $AB = I$ and $AKB = \ldots = AK^tB = O$ can be rewritten as

$$\vec{a}_i^T \vec{b} = \begin{cases} 1 & (\vec{b} = \vec{b}_i) \\ 0 & (\vec{b} \in V \setminus \{\vec{b}_i\}) \end{cases} \quad (i = 1, \ldots, Q).$$

Therefore, $\vec{b}_i$ cannot be linearly expressed by $V \setminus \{\vec{b}_i\}$.

We claim that $V \setminus \{\vec{b}_1, \ldots, \vec{b}_Q\} = \{\vec{k}^j \odot \vec{b}_i\}_{j=1\ldots,t,i=1,\ldots,Q}$ is linearly dependent. Otherwise since $\vec{b}_1$ cannot be linearly expressed by $V \setminus \{\vec{b}_1\}$, it cannot be linearly expressed by its subset $V \setminus \{\vec{b}_1, \ldots, \vec{b}_Q\}$ as well; therefore, adding $\vec{b}_1$ to $V \setminus \{\vec{b}_1, \ldots, \vec{b}_Q\}$, that is, $V \setminus \{\vec{b}_2, \ldots, \vec{b}_Q\}$, is still linearly independent. Similar with above, we can add $\vec{b}_2, \ldots, \vec{b}_Q$ to the set and remain its linear independency, i.e., V is also linearly independent. But this is impossible according to Lemma 2.

Since $\{\vec{k}^j \odot \vec{b}_i\}_{j\in[t],i\in[Q]}$ is linearly dependent, there exist x_{ij} $(j = 1, \ldots, t,\ i = 1, \ldots, Q)$, at least one of which is non-zero, such that

$$\sum_{j=1}^{t} \sum_{i=1}^{Q} x_{ij} \vec{k}^j \odot \vec{b}_i = \vec{0}.$$

Because none of $\vec{k}$'s entries is 0, we can derive that

$$\sum_{j=1}^{t} \sum_{i=1}^{Q} x_{ij} \vec{k}^{j-1} \odot \vec{b}_i = \vec{0},$$

i.e.,

$$\sum_{i=1}^{Q} x_{i1} \vec{b}_i + \sum_{j=1}^{t-1} \sum_{i=1}^{Q} x_{i,j+1} \vec{k}^j \odot \vec{b}_i = \vec{0}.$$

Recall that $\vec{b}_i$ cannot be linearly expressed by $V \setminus \{\vec{b}_i\}$, so $x_{11} = \ldots = x_{Q1} = 0$. We get that

$$\sum_{j=1}^{t-1} \sum_{i=1}^{Q} x_{i,j+1} \vec{k}^j \odot \vec{b}_i = 0.$$

Using the same steps above, we can see $x_{12} = \ldots = x_{Q2} = 0$, then $x_{13} = \ldots = x_{Q3} = 0$, $\ldots$, finally $x_{1Q} = \ldots = x_{QQ} = 0$. That is, all x_{ij}'s ($j = 1, \ldots, t$, $i = 1, \ldots, Q$) are 0, which contradicts the claim above. □

Theorem 11. (generic group hardness of $(0, t, n, Q+1, Q)$-T-OMDH) *Let $\mathbb{G}$ be a generic group of prime order m. We use $\xi(a)$ for $a \in \mathbb{Z}_m$ to denote (the encoding of) elements in $\mathbb{G}$, where $\xi(\cdot)$ is a random 1-1 function mapping $\mathbb{Z}_m$ to bitstrings of sufficient size. Let $\mathcal{A}$ be an algorithm which can query the following two oracles:*

- *Group operation oracle, which on input $(\xi(a_1), \xi(a_2))$ and an operation, either $+$ or $-$, respectively outputs $\xi(a_1 + a_2)$ or $\xi(a_1 - a_2)$;*
- *Oracle $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$, where $p(\cdot)$ is a t-degree polynomial over $\mathbb{Z}_m$, which on input $(k, \xi(a))$ for $k \in \{1, \ldots, n\}$ outputs $\xi(a \cdot p(k))$.*

Let $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)}(t, n, Q, r, m)$ be the probability that $\mathcal{A}(\xi(1), \xi(u_1), \ldots, \xi(u_{Q+1}))$ outputs $(\xi(u_1 \cdot p(0)), \ldots, \xi(u_{Q+1} \cdot p(0)))$ after making r group operation queries and q_i queries to $\mathsf{T\text{-}OMDH}_p(i, \cdot)$ such that $C_{t+1}(\vec{q}) \leq Q$. Then

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)}(t, n, Q, r, m) \leq \frac{(w+1)(w+Q+r+2)^2 + 4}{2m},$$

where $w = W(\vec{q})$ (i.e., w is the total number of queries to the T-OMDH oracle), and the probability goes over the random choice of t-degree polynomial p, the randomness of $\mathcal{A}$ and the randomness of oracle $\xi(\cdot)$.

Proof. The proof goes by construction of an algorithm $\mathcal{B}$ which simulates the real

challenger while interacting with $\mathcal{A}$. $\mathcal{B}$ maintains a list $T := \{(F_s, \xi_s)\}_{s=1,\ldots,\sigma}$, where $F_s(U_1, \ldots, U_{Q+1}, A_0, A_1, \ldots, A_t)$ is a polynomial of degree at most w, and ξ_s's are random distinct elements in $\mathbb{G}$. At the beginning, $\mathcal{B}$ sets $\sigma := Q + 2$ and initializes T by setting $F_1 := 1, F_2 := u_1, \ldots, F_{Q+2} := u_{Q+1}$, and picks $\xi_1, \ldots, \xi_{Q+2}$ as random distinct elements in $\mathbb{G}$ and $a_0, a_1, \ldots, a_t \leftarrow_{\mathrm{R}} \mathcal{F}$. $\mathcal{B}$ sends $\xi_1, \ldots, \xi_{Q+2}$ to $\mathcal{A}$ as $\xi(1), \xi(u_1), \ldots, \xi(u_{Q+1})$. Then $\mathcal{A}$ can make the following two types of queries to $\mathcal{B}$ (we assume that $\mathcal{A}$ only makes oracle queries on values that are previously obtained from $\mathcal{B}$):

- Group operation query: $\mathcal{A}$ inputs two indices s_1 and s_2, as well as an operation (either a multiplication or a division). $\mathcal{B}$ computes $F_{\sigma+1} := F_{s_1} + F_{s_2}$ if the operation is a multiplication or $F_{\sigma+1} := F_{s_1} - F_{s_2}$ if the operation is a division. If $\exists\ t \leq \sigma$ s.t. $F_s = F_{\sigma+1}$, then $\mathcal{B}$ outputs ξ_t to $\mathcal{A}$. Otherwise $\mathcal{B}$ picks a group element $\xi_{\sigma+1}$ which is different from ξ_i for all $i \leq \sigma$, outputs $\xi_{\sigma+1}$ to $\mathcal{A}$, and sets $\sigma{+}{+}$.
- $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$ oracle query: $\mathcal{A}$ inputs a $k \in [n]$ and an index $s \in \{1, \ldots, \sigma\}$. Then $\mathcal{B}$ sets $F_{\sigma+1} := (\vec{L}_k^T \vec{a}) \cdot F_s$ and $\xi_{\sigma+1}$ to a random value which is different from ξ_s for all $s \leq \sigma$, outputs $\xi_{\sigma+1}$ to $\mathcal{A}$, and sets $\sigma{+}{+}$, where $\vec{L}_k = [1, k, \ldots, k^t]^T$ and $\vec{a} = [a_0, a_1, \ldots, a_t]^T$.

$\mathcal{A}$ finally outputs $(F_{s_1}, \ldots, F_{s_{Q+1}})$, and it wins if $F_{s_i} = u_i \cdot a_0$ for all $i \in [Q + 1]$.

Now we analyze the probability that $\mathcal{A}$ succeeds for a random assignment of $(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t)$. First, note that the output of $\mathcal{A}$ comes from the two types of oracle queries listed above; therefore, for every $s \in \{s_1, \ldots, s_{Q+1}\}$, F_s is a linear combination of $v_1, \ldots, v_w$, $u_1, \ldots, u_{Q+1}$ and 1, where v_i $(i = 1, \ldots, w)$ is the value $\mathcal{A}$ obtains from the ith $\mathsf{T\text{-}OMDH}_p(\cdot, \cdot)$ oracle query. That is,

$$F_s = \sum_{i=1}^{w} \alpha_i^{(s)} v_i + \sum_{i=0}^{Q+1} \gamma_i^{(s)} u_i,$$

where

$$v_i = \sum_{Z \subseteq [i] s.t. i \in Z} \left[\left(\sum_{j=0}^{Q+1} \beta_{jZ}^{(i)} u_j \right) \prod_{l \in Z} (\vec{L}_{kl}^T \vec{a}) \right]$$

(we set $u_0 = 1$ in the two equations above), where α_i's, γ_i's and β_{jZ}'s are all elements in $\mathbb{Z}_m$ specified by $\mathcal{A}$. (Suppose that in the ith $\mathsf{T\text{-}OMDH}_p(\cdot,\cdot)$ oracle query, $\mathcal{A}$'s second input is k_i; then $\vec{L}_{k_i}^T \vec{a}$ must appear in the output. That is why $i \in Z$ holds in expression of w_i.)

Recall that $\mathcal{A}$ wins if and only if $F_{s_i} = u_i \cdot a_0$ for all $i = 1, \ldots, Q+1$. Suppose that there exists an $i \in \{1, \ldots, Q+1\}$ such that $\deg(F_{s_i}) > 1$ but $\mathcal{A}$ still wins, i.e., $F_{s_i} = u_i \cdot a_0$ (we view F_{s_i} as a polynomial of a_0 here). Since $\deg(u_i \cdot a_0) = 1$, the only possibility that this occurs is that the polynomials F_{t_i} and $u_i \cdot a_0$ evaluates to the same value for random $u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t$. Also note that $\deg(F_{s_i}) \leq w$, thus $\deg(F_{s_i} - u_i \cdot a_0) \leq w$; therefore, if the above occurs for a certain i, either (i) $1 \leq \deg(F_{s_i} - u_i \cdot a_0) \leq w$, and random a_0 is a solution of $F_{s_i} - u_i \cdot a_0$, or (ii) $F_{s_i} - u_i \cdot a_0$ is the zero polynomial for fixed $u_1, \ldots, u_{Q+1}$ chosen from random. The probability of (i) is at most w/m, while the probability of (ii) is at most $1/p$. Since there are $Q+1$ possible values of i, the probability that there exists an $i \in \{1, \ldots, Q+1\}$ such that $\deg(F_{s_i}) > 1$ but $\mathcal{A}$ still wins is at most $(w+1)(Q+1)/m$.

Next consider the case where $\deg(F_{s_i}) \leq 1$ for all $i = 1, \ldots, Q+1$. Let v_i' be v_i with all terms whose degree greater than 1 eliminated, that is, v_i' only remains the single term in v_i where $Z = \{i\}$, i.e.,

$$v_i' = \left(\sum_{j=0}^{Q+1} \beta_{j\{1\}}^{(i)} u_j \right) (\vec{L}_{k_i}^T \vec{a}).$$

Then

$$F_s = \sum_{i=1}^{w} \alpha_i^{(s)} v_i' + \sum_{i=0}^{w+1} \gamma_i^{(s)} u_i.$$

We rewrite the expression of F_s in matrix form below. Note that since $\deg(v_i') \leq 1$, all $\beta_{jZ}^{(i)}$'s

for $Z \neq \{1\}$ do not appear in the expression of v'_i; therefore, we denote $\beta^{(i)}_{j\{1\}}$ as $\beta^{(i)}_j$:

$$\begin{bmatrix} F_{s_1} \\ \vdots \\ F_{s_{Q+1}} \end{bmatrix} = A \begin{bmatrix} v'_1 \\ \vdots \\ v'_w \end{bmatrix} + C\vec{u} + \begin{bmatrix} \gamma_0^{(s_1)} \\ \vdots \\ \gamma_0^{(s_{Q+1})} \end{bmatrix}, \tag{A.1}$$

$$\begin{bmatrix} v'_1 \\ \vdots \\ v'_w \end{bmatrix} = (B\vec{u} + \vec{b}_0) \odot \begin{bmatrix} \vec{L}_{k_1}^T \vec{a} \\ \vdots \\ \vec{L}_{k_w}^T \vec{a} \end{bmatrix}, \tag{A.2}$$

where

$$A = \begin{bmatrix} \alpha_1^{(s_1)} & \dots & \alpha_w^{(s_1)} \\ \vdots & & \vdots \\ \alpha_1^{(s_{Q+1})} & \dots & \alpha_Q^{(s_{Q+1})} \end{bmatrix}_{(Q+1)\times w}, B = \begin{bmatrix} \beta_1^{(1)} & \dots & \beta_{Q+1}^{(1)} \\ \vdots & & \vdots \\ \beta_1^{(w)} & \dots & \beta_{Q+1}^{(w)} \end{bmatrix}_{w\times(Q+1)},$$

$$C = \begin{bmatrix} \gamma_1^{(1)} & \dots & \gamma_{Q+1}^{(1)} \\ \vdots & & \vdots \\ \gamma_1^{(Q+1)} & \dots & \gamma_{Q+1}^{(Q+1)} \end{bmatrix}_{(Q+1)\times(Q+1)}, \vec{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_{Q+1} \end{bmatrix}, \vec{b}_0 = \begin{bmatrix} \beta_0^{(1)} \\ \vdots \\ \beta_0^{(w)} \end{bmatrix}.$$

Let $\vec{b} = B\vec{u} + \vec{b}_0$. Substituting (A.2) into (A.1), we get that

$$\begin{bmatrix} F_{s_1} \\ \vdots \\ F_{s_{Q+1}} \end{bmatrix} = A \left(\vec{b} \odot \begin{bmatrix} \vec{L}_{k_1}^T \vec{a} \\ \vdots \\ \vec{L}_{k_w}^T \vec{a} \end{bmatrix} \right) + C\vec{u} + \begin{bmatrix} \gamma_0^{(s_1)} \\ \vdots \\ \gamma_0^{(s_{Q+1})} \end{bmatrix}.$$

Note that

$$\begin{bmatrix} \vec{L}_{k_1}^T \vec{a} \\ \vdots \\ \vec{L}_{k_w}^T \vec{a} \end{bmatrix} = \begin{bmatrix} a_0 + a_1 k_1 + \ldots + a_t k_1^t \\ \vdots \\ a_0 + a_w k_w + \ldots + a_t k_w^t \end{bmatrix} = a_0 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} + a_1 \begin{bmatrix} k_1 \\ \vdots \\ k_w \end{bmatrix} + \ldots + a_t \begin{bmatrix} k_1^t \\ \vdots \\ k_w^t \end{bmatrix},$$

according to the properties of Hadamard product:

$$\begin{aligned} \begin{bmatrix} F_{s_1} \\ \vdots \\ F_{s_{Q+1}} \end{bmatrix} &= A(a_0 I\vec{b} + a_1 K\vec{b} + \ldots + a_t K^t \vec{b}) + C\vec{u} + \begin{bmatrix} \gamma_0^{(s_1)} \\ \vdots \\ \gamma_0^{(s_{Q+1})} \end{bmatrix} \\ &= a_0 A\vec{b} + a_1 AK\vec{b} + \ldots + a_t AK^t \vec{b} + C\vec{u} + \begin{bmatrix} \gamma_0^{(s_1)} \\ \vdots \\ \gamma_0^{(s_{Q+1})} \end{bmatrix}, \end{aligned}$$

$$\text{where} \quad K = \begin{bmatrix} k_1 & & \\ & \ddots & \\ & & k_w \end{bmatrix}_{w \times w}.$$

View the right side as a linear function of $a_0, a_1 \ldots, a_t$, that is, consider the right side for some fixed random $u_1, \ldots, u_{w+1}$. If $\mathcal{A}$ wins, then

$$\begin{bmatrix} F_{s_1} \\ \vdots \\ F_{s_{Q+1}} \end{bmatrix} = a_0 \vec{u},$$

comparing the two equations, we have

$$A\vec{b} = \vec{u}, AK\vec{b} = \ldots = AK^t \vec{b} = \vec{0},$$

except for the case where the two linear functions evaluate to the same value for random $a_0, a_1, \ldots, a_t \in \mathbb{Z}_m$. The probability that this occurs is at most $1/p$.

Substituting the definition of $\vec{b}$ back, we get that

$$AB\vec{u} + A\vec{b_0} = \vec{u}, \text{ i.e. } (AB - I)\vec{u} + A\vec{b_0} = \vec{0},$$

$$AKB\vec{u} + AK\vec{b_0} = \ldots = AK^tB\vec{u} + AK^t\vec{b_0} = \vec{0}.$$

According to Lemma 3, there is at least one of $AB - I, AKB, \ldots, AK^tB$ which is not O. Then there exists at least one row of one of the matrices above which is not $\vec{0}$; let it be the i-th row of the t_0-th matrix ($t_0 \in \{0, \ldots, t\}$). Denote that row as $\vec{z}^T$. Then we have

$$\vec{z}^T\vec{u} + AK^{t_0}\beta_0^{(i)} = \vec{0}$$

for random $u_1, \ldots, u_{Q+1} \in \mathbb{Z}_m$. The probability that this occurs is at most $1/p$.

In sum, we have proved that the probability that $\mathcal{A}$ succeeds in the game interacting with $\mathcal{B}$ is at most

$$\frac{(w+1)(Q+1)}{m} + \frac{1}{m} + \frac{1}{m} = \frac{(w+1)(Q+1)+2}{m}.$$

The difference between $\mathcal{A}$'s views in the interaction with $\mathcal{B}$ and with the real challenger of the T-OMDH assumption in the generic group model appears when there exist s_1 and s_2 such that $F_{s_1}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t) = F_{s_2}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t)$ for random $u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t$, but the polynomials F_{s_1} and F_{s_2} are not the same. There are $\binom{\sigma}{2}$ possible (s_1, s_2) pairs, and for each such pair, the probability that $F_{s_1}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t) = F_{s_2}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t)$ is at most $(w+1)/m$. Thus, the probability that the event above occurs is at most $\binom{\sigma}{2} \cdot (w+1)/m$. Also note that each time $\mathcal{A}$ queries one of the two oracles, σ either remains the same or increases by

1; there are at most $r + w$ such queries, and $\sigma = Q + 2$ at the beginning. So $\sigma \leq w + Q + r + 2$.

Therefore, we have proved that the probability that $\mathcal{A}$ breaks the T-OMDH assumption where $t' = 0$ in the generic group model is at most

$$\frac{(w+1)(Q+1)+2}{m} + \binom{w+Q+r+2}{2} \cdot \frac{w+1}{m} \leq \frac{(w+1)(w+Q+r+2)^2+4}{2m}.$$

□

Next we consider the general case, i.e. $t' > 0$ and $\mathsf{Bad} \neq \varnothing$:

Theorem 12. (generic group hardness of $(t', t, n, Q+1, Q)$-T-OMDH) *Let $\mathbb{G}$ and $\xi(\cdot)$ be the same with those in Theorem 11. Let Bad be any t-element subset of $\{1, \ldots, n\}$. $\mathcal{A}$ is given $\xi(1), \xi(u_1), \ldots, \xi(u_{Q+1})$. Then $\mathcal{A}$ outputs a map $F : \mathsf{Bad} \to \mathbb{Z}_m$. After that, $\mathcal{A}$ can query the group operation oracle and the T-OMDH oracle as in Theorem 11, with the exception that $p(\cdot)$ is a t-degree polynomial over $\mathbb{Z}_m$ such that $p(\alpha) = F(\alpha)$ for all $\alpha \in \mathsf{Bad}$. $\mathcal{A}$ wins if it outputs $(\xi(u_1 \cdot p(0)), \ldots, \xi(u_{Q+1} \cdot p(0)))$.*

The bound on adversarial advantage, i.e., on probability that $\mathcal{A}(\xi(1), \xi(u_1), \ldots, \xi(u_{Q+1}))$ outputs $(\xi(u_1 \cdot p(0)), \ldots, \xi(u_{Q+1} \cdot p(0)))$ after making r group operation queries and q_i queries to $\mathsf{T\text{-}OMDH}_p(i, \cdot)$ for $i \notin \mathsf{Bad}$ such that $C_{t-t'+1}(\vec{q}) \leq Q$, is exactly the same as the bound on the adversarial advantage in Theorem 11.

Proof. The proof is an extension of that of Theorem 11, so we only provide a sketch. At the beginning of the simulated game, $\mathcal{A}$ chooses $F(\alpha) = \vec{L}_i^T \vec{a}$ ($\alpha \in \mathsf{Bad}$). Since $\mathcal{A}$ knows t' of $\vec{L}_i^T \vec{a}$'s, there are $t + 1$ variables and t' linear equations (which are linearly independent), so there are $t - t' + 1$ free variables; in particular, $a_0, \ldots, a_{t-t'}$ are still independently random

from $\mathcal{A}$'s view. The whole argument holds until the following step:

$$A\vec{b} = \vec{u}, AK\vec{b} = \ldots = AK^t\vec{b} = \vec{0},$$

where t should be replaced by $t - t'$. After that, the argument still holds if we replace t by $t - t'$. □

Theorem 13. *Suppose furthermore that $\mathbb{G}$ is a gap group, that is, $\mathcal{A}$ can make the following type of oracle queries in addition:*

- *DDH oracle, which on input $\xi(a_1), \xi(a_2), \xi(a_3), \xi(a_4)$ which are different with each other, outputs 1 if $a_1 a_4 = a_2 a_3$, and 0 otherwise. $\mathcal{A}$ can make such queries at most q times.*

Then the bound on adversarial advantage of Theorem 12 still holds, with an upper-bound modified to $\mathbf{Adv}_{\mathcal{A}} \leq \frac{(w+1)(w+Q+r+2)^2+2(2w+1)q+4}{2m}$.

Proof. We construct an algorithm $\mathcal{B}'$ which is exactly the same with $\mathcal{B}$ in the previous proof, except that $\mathcal{A}$ can make the following oracle queries to $\mathcal{B}'$ as well:

- DDH oracle query: $\mathcal{A}$ inputs four different indices s_1, s_2, s_3 and s_4. $\mathcal{B}'$ outputs 1 to $\mathcal{A}$ if $F_{s_1} F_{s_4} = F_{s_2} F_{s_3}$, and 0 otherwise.

Now the difference between $\mathcal{A}$'s views in the interaction with $\mathcal{B}$ and in the interaction with the real challenger appears additionally when there are four different s_1, s_2, s_3, s_4 such that

$$\begin{aligned} &F_{s_1}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t) F_{s_4}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t) \quad = \\ &\quad F_{s_2}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t) F_{s_3}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t) \end{aligned}$$

for random $u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t$, but the polynomials $F_{s_1}F_{s_4}$ and $F_{s_2}F_{s_3}$ are not the same. Note that $\deg(F_{s_1}F_{s_4} - F_{s_2}F_{s_3}) < 2w$, and there are at most q such polynomials evaluated, so the probability that the above event occurs is at most $(2w+1)q/m$. All other proof steps remain as in the proof of Theorem 12. $\square$